ETH zürich

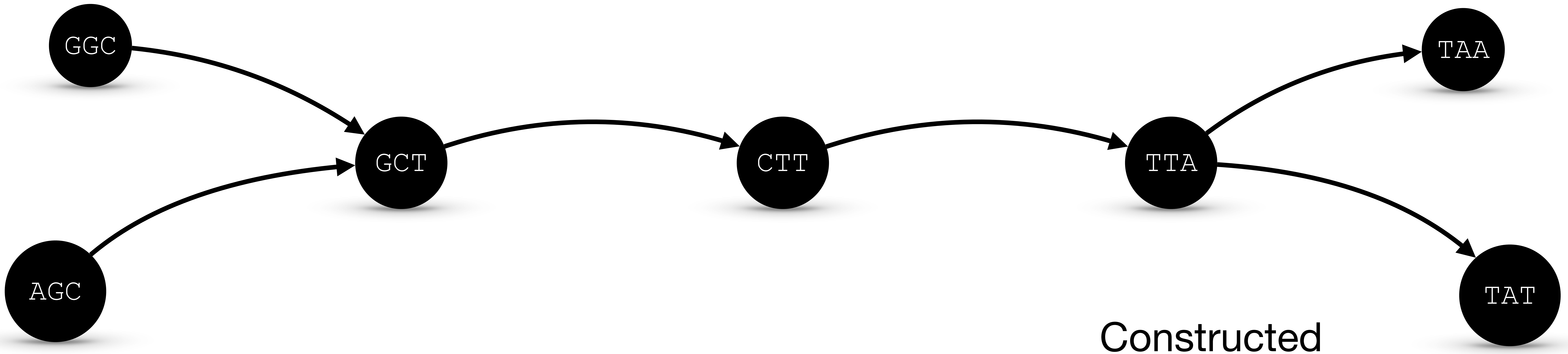
Topology-based Sparsification of Graph Annotations

Daniel Danciu*, Mikhail Karasikov*, Harun Mustafa, André Kahles, Gunnar Rätsch

ISMB/ECCB 2021

Background

Annotated de Bruijn Graphs



Constructed
from sequences

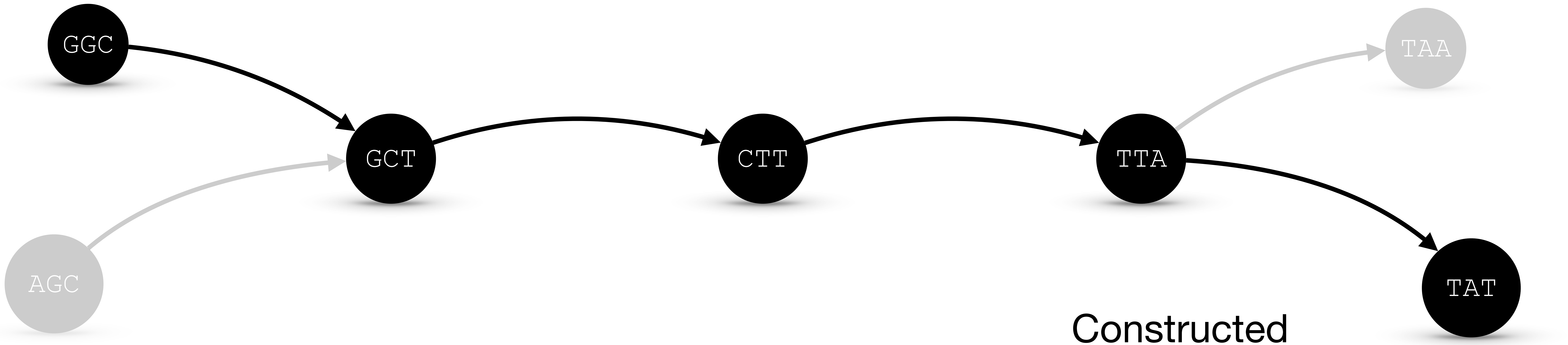
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed
from sequences

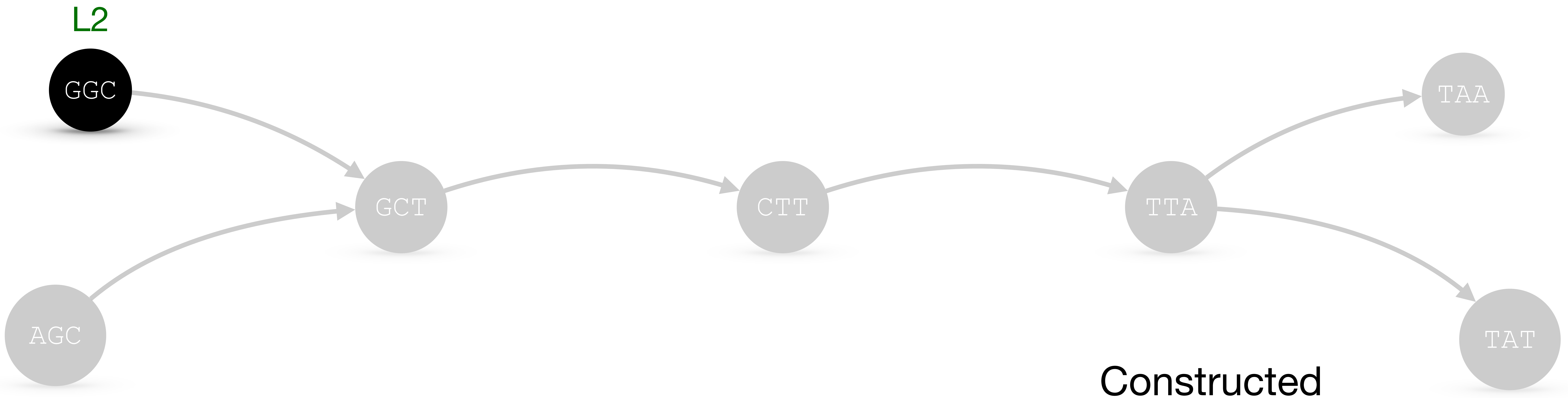
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed
from sequences

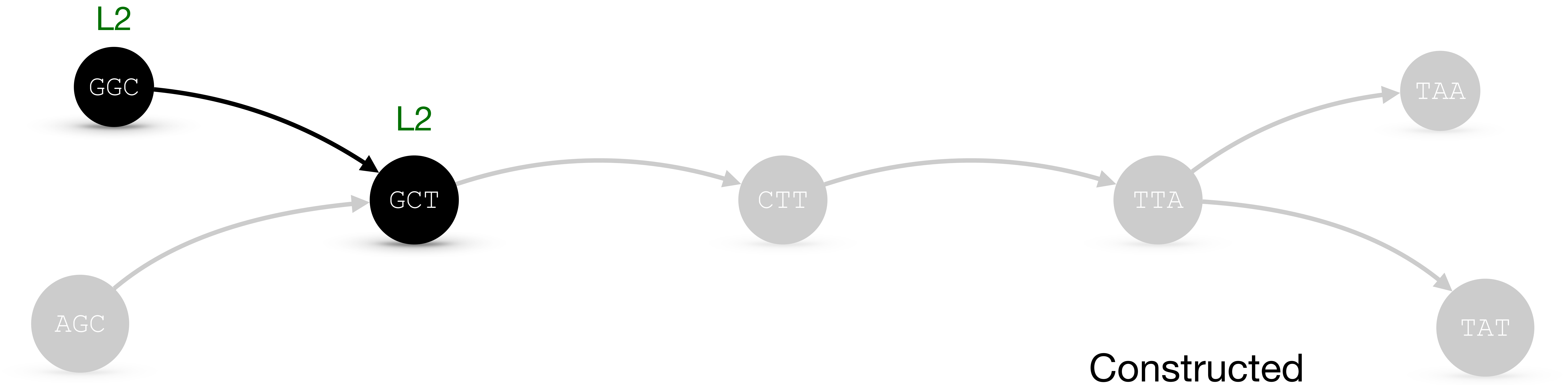
L1: AGCTTAA

L2: $\boxed{\text{GGC}}$TTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed from sequences

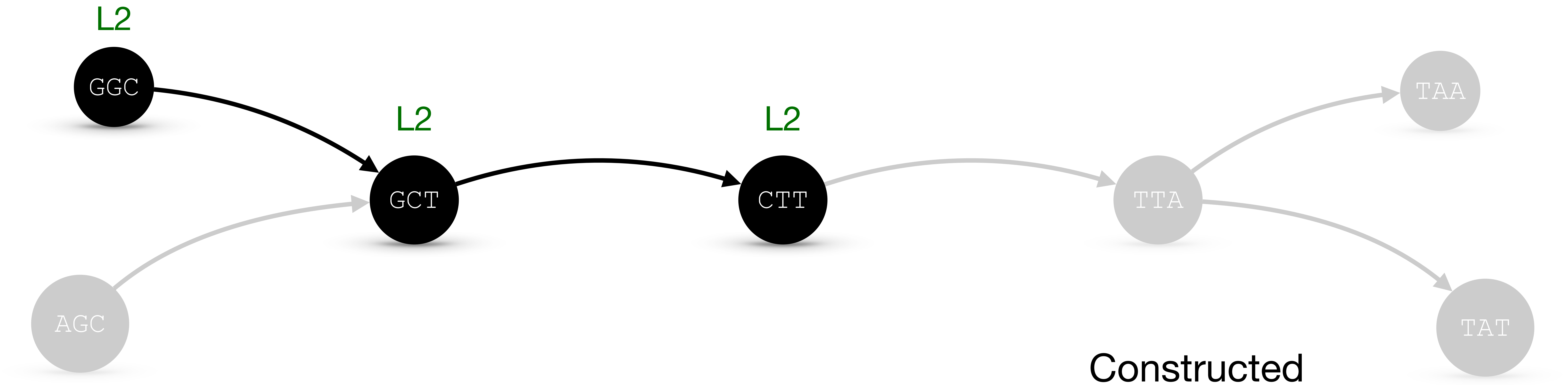
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed
from sequences

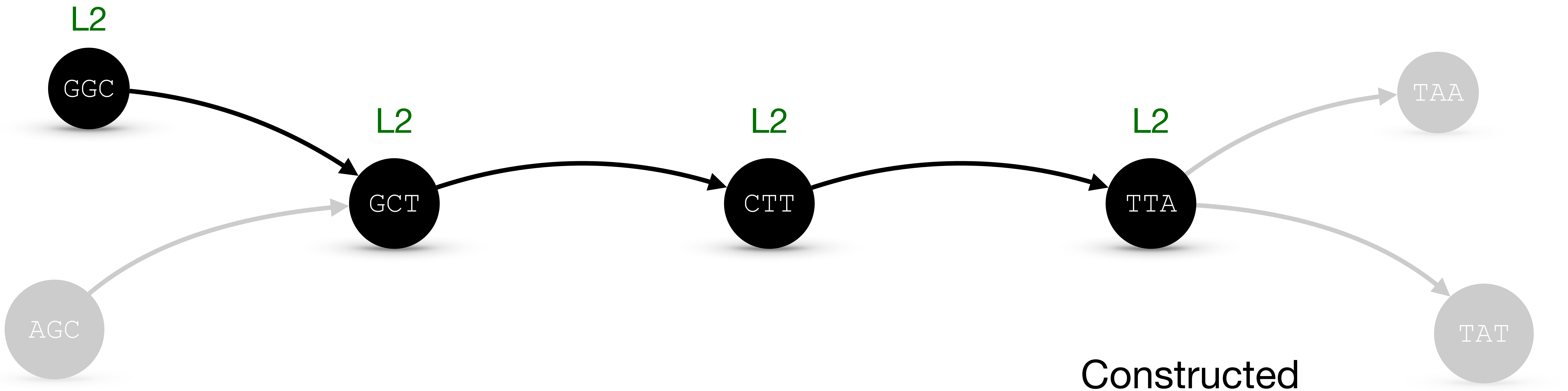
L1: AGCTTAA

L2: GG$\boxed{\text{CTT}}$AT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed
from sequences

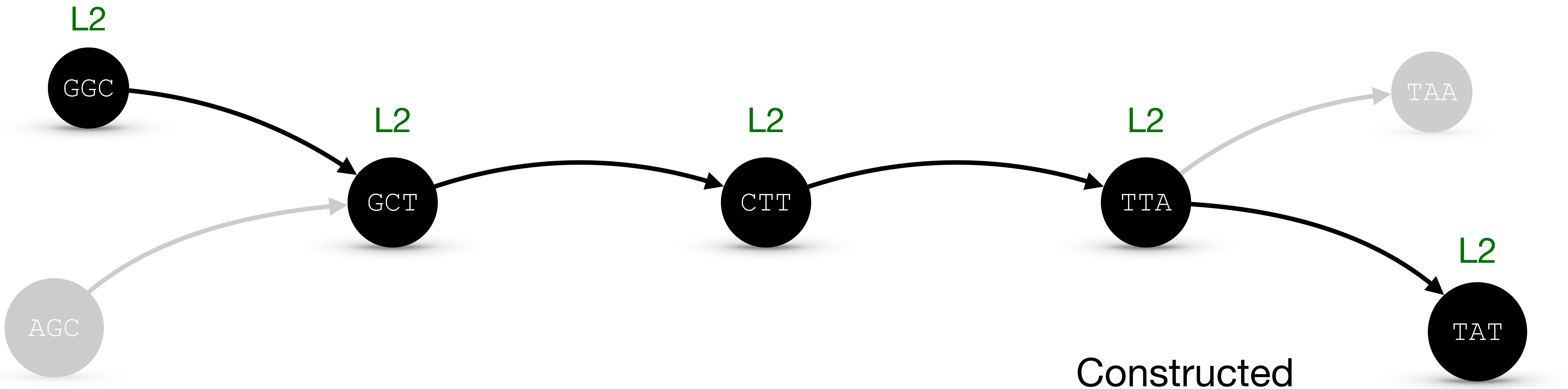
L1: AGCTTAA

L2: GGC$\boxed{\text{TTA}}$T

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed
from sequences

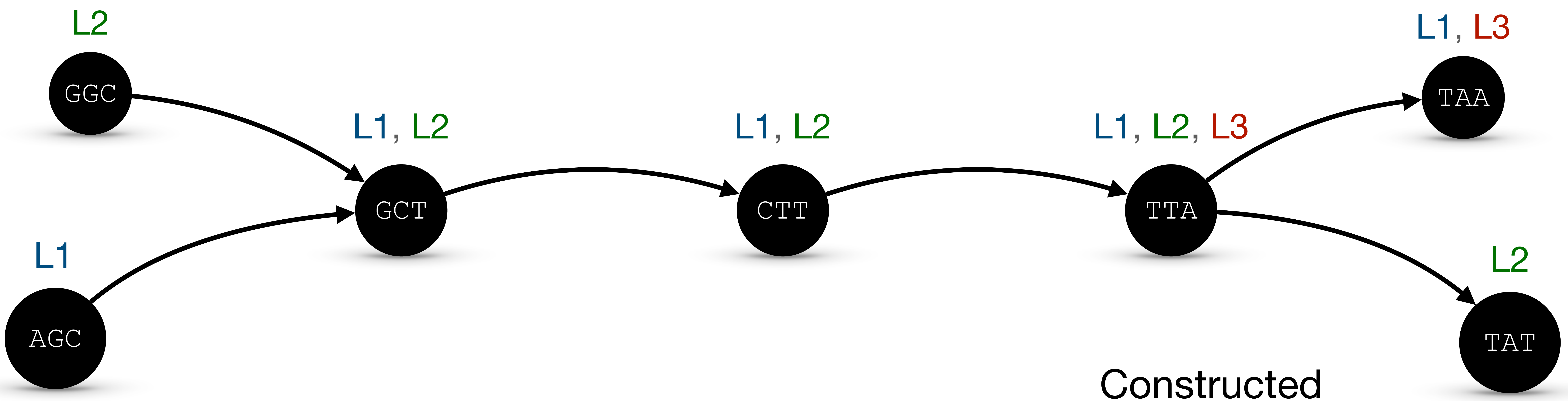
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Constructed from sequences

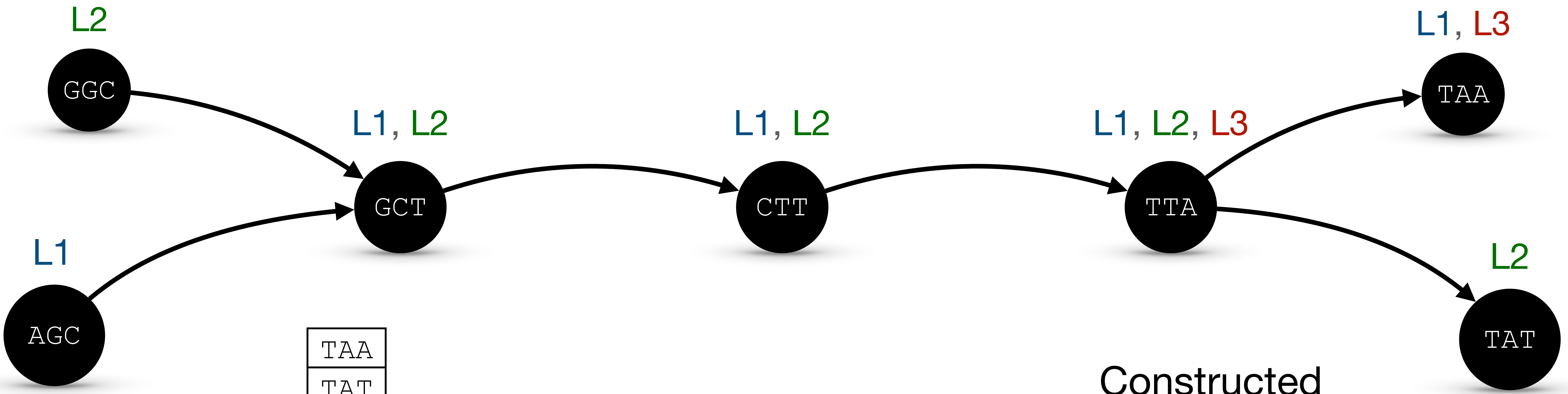
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



TAA
TAT
GCT
AGC
GGC
CTT
TTA

k-mer dictionary
(de Bruijn Graph)

Constructed from sequences

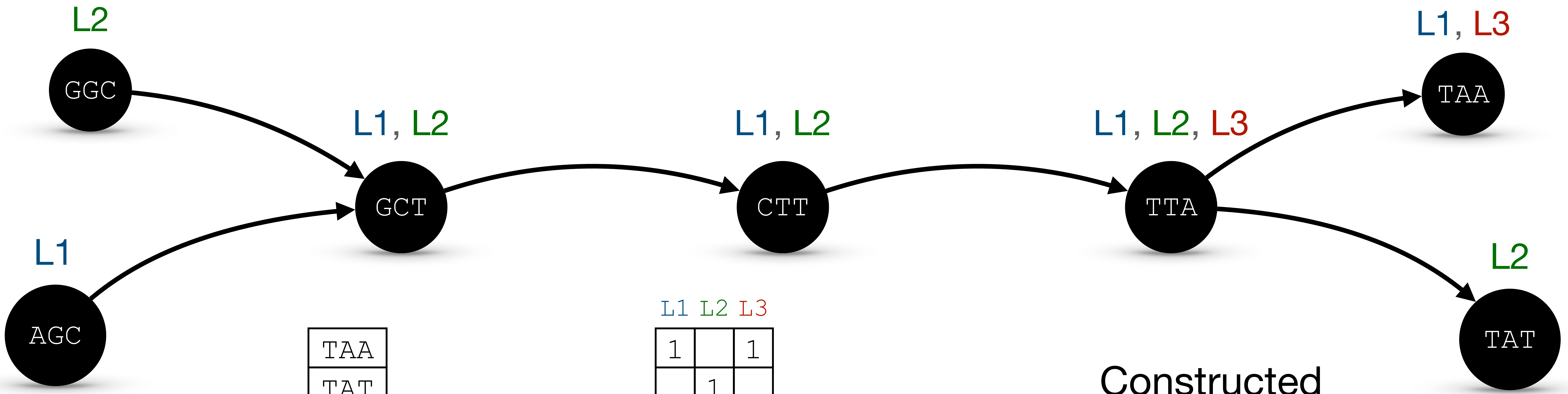
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



TAA
TAT
GCT
AGC
GGC
CTT
TTA

k-mer dictionary
(de Bruijn Graph)

L1	L2	L3
1		1
	1	
1	1	
1		
	1	
1	1	
1	1	1

Graph annotation
(labeling)

Constructed from sequences

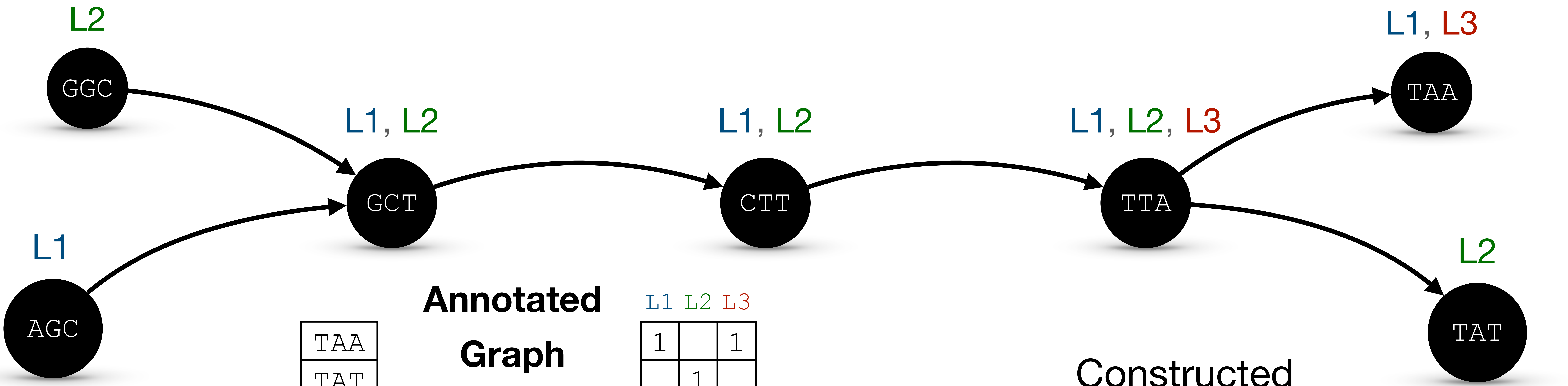
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Annotated Graph

k-mer	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

k-mer dictionary (de Bruijn Graph) | Graph annotation (labeling)

Constructed from sequences

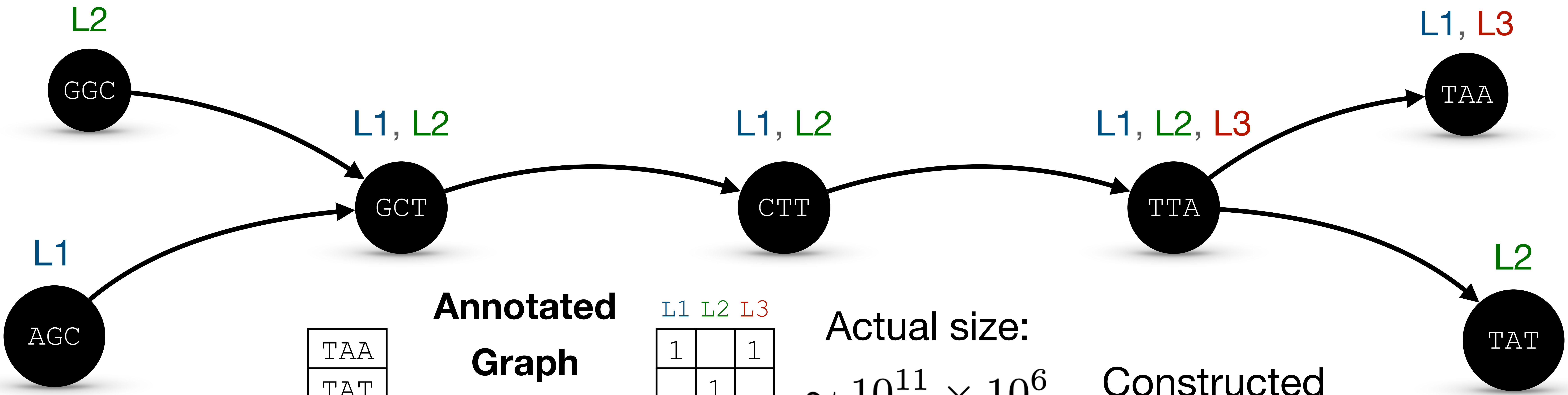
L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Annotated de Bruijn Graphs



Annotated Graph

TAA
TAT
GCT
AGC
GGC
CTT
TTA

k-mer dictionary
(de Bruijn Graph)

L1	L2	L3
1		1
	1	
1	1	
1		
	1	
1	1	
1	1	1

Graph annotation
(labeling)

Actual size:

$\sim 10^{11} \times 10^{6}$

Constructed from sequences

L1: AGCTTAA

L2: GGCTTAT

L3: TTAA

Background

Graph Annotation Representations

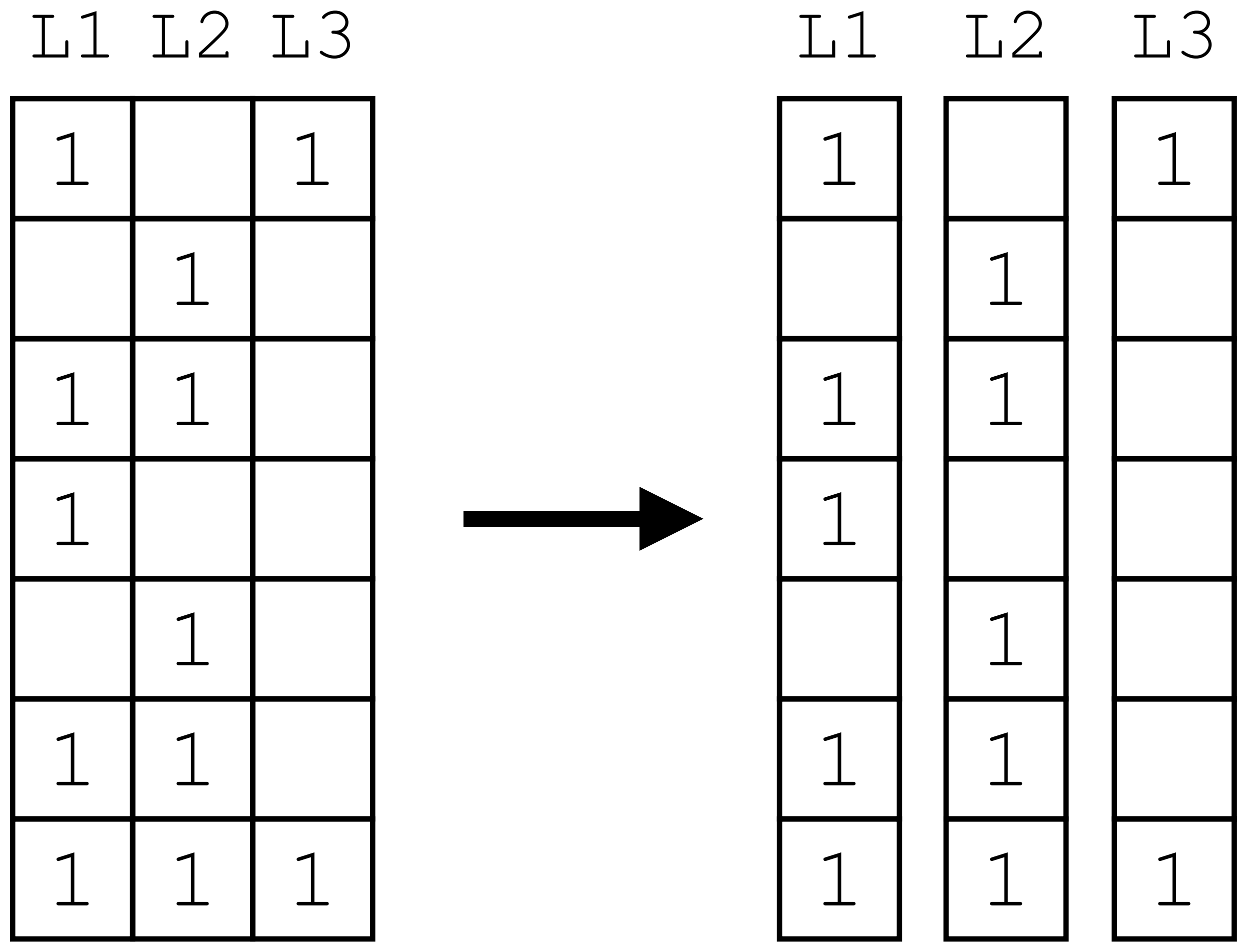
	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

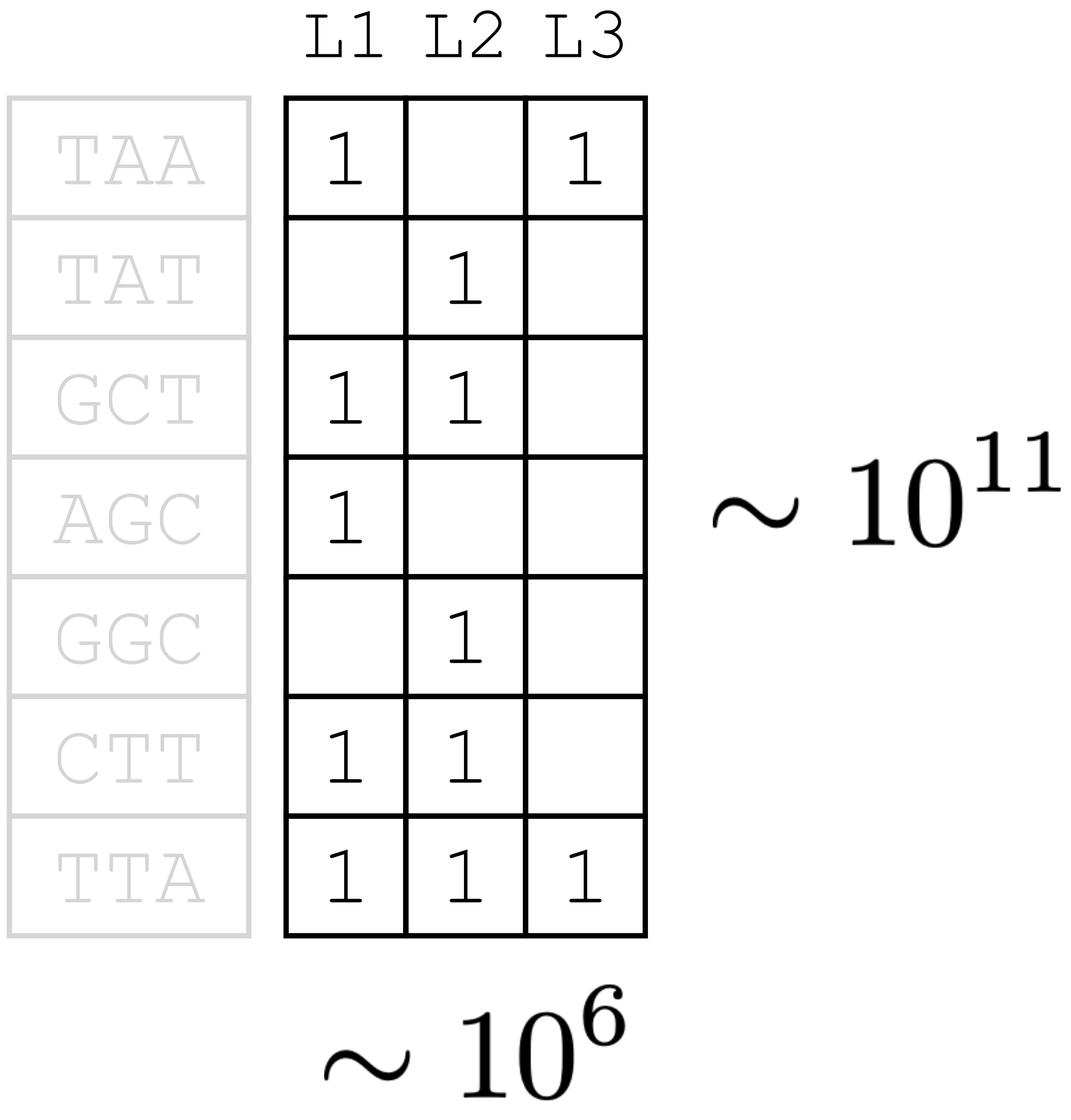
$\sim 10^{11}$

$\sim 10^{6}$

Background
Graph Annotation Representations

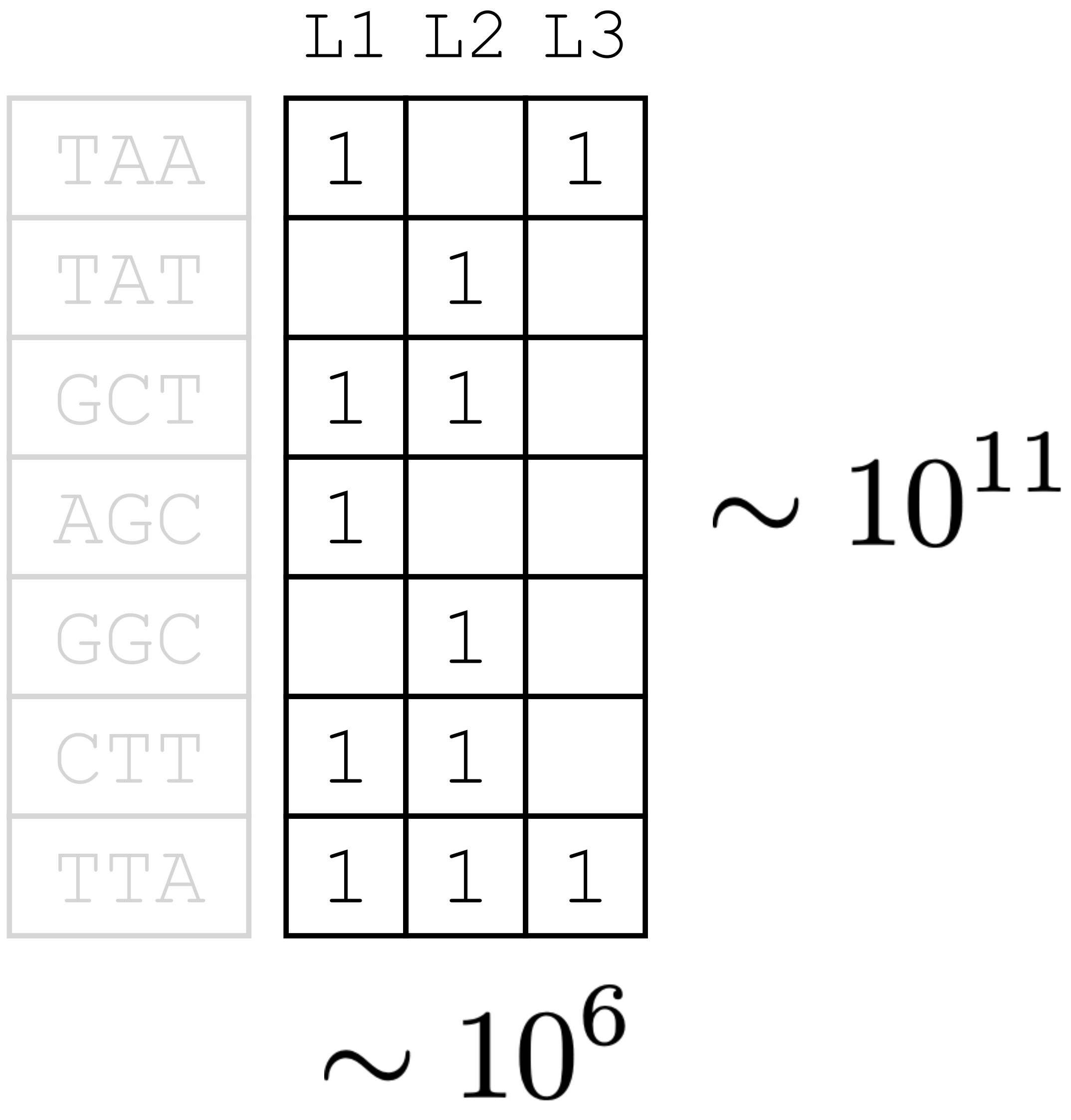
1. Column-major sparse representation



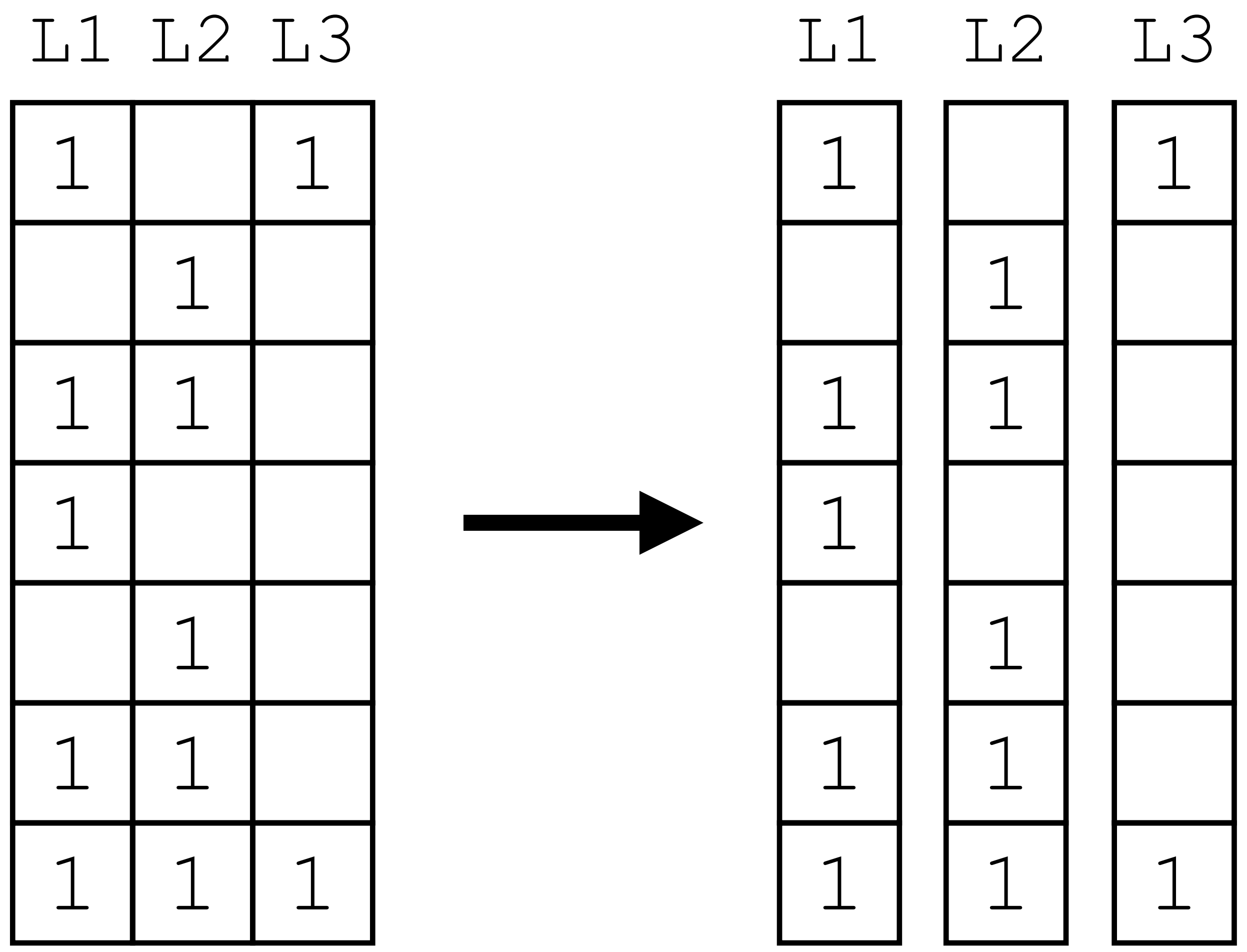


Background

Graph Annotation Representations



1. Column-major sparse representation



Columns are stored independently as compressed bitmaps
(e.g. sd_vector [Okanohara et al., 2007])

Background

Graph Annotation Representations

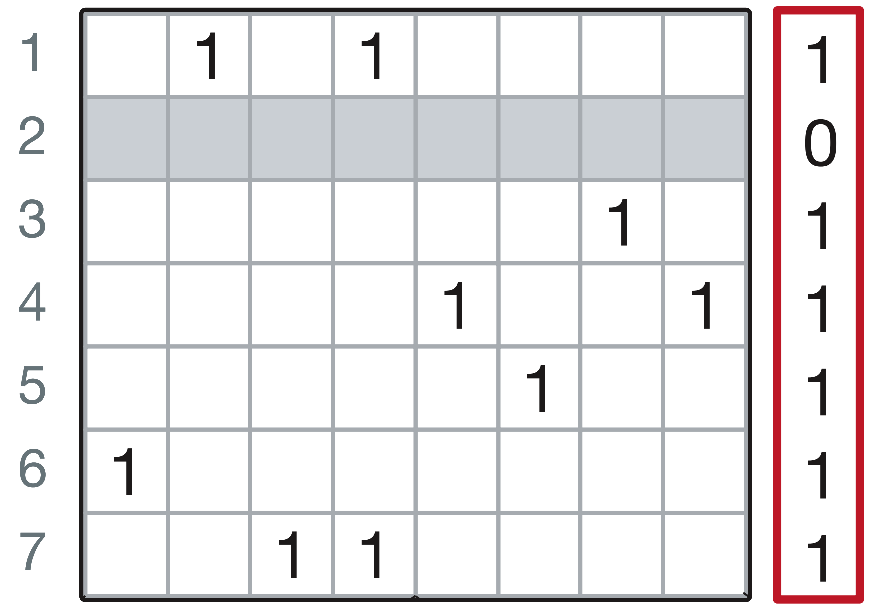
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**

1		1		1				
2								
3							1	
4					1			1
5						1		
6	1							
7			1	1				

Background

Graph Annotation Representations

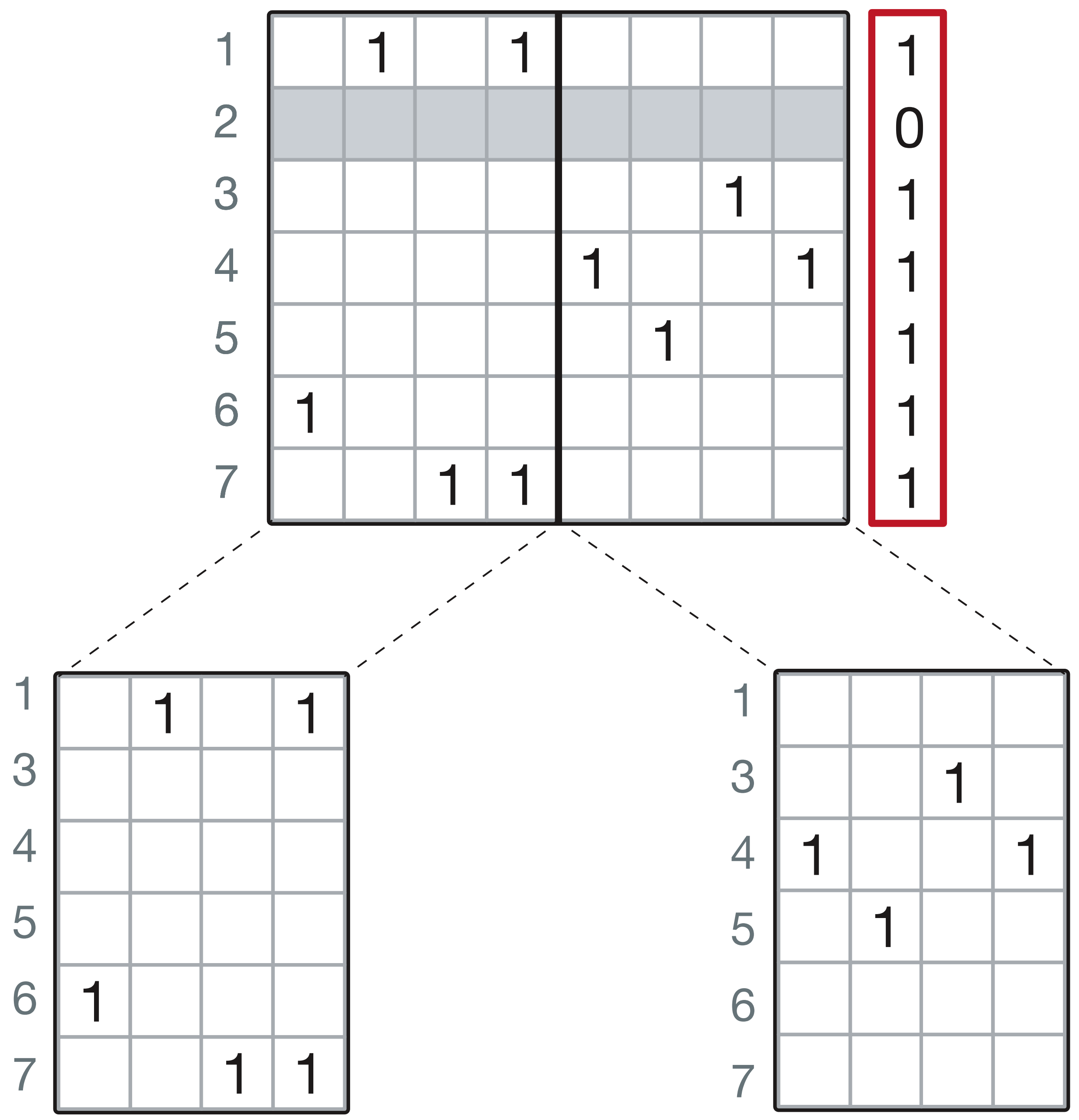
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**



Background

Graph Annotation Representations

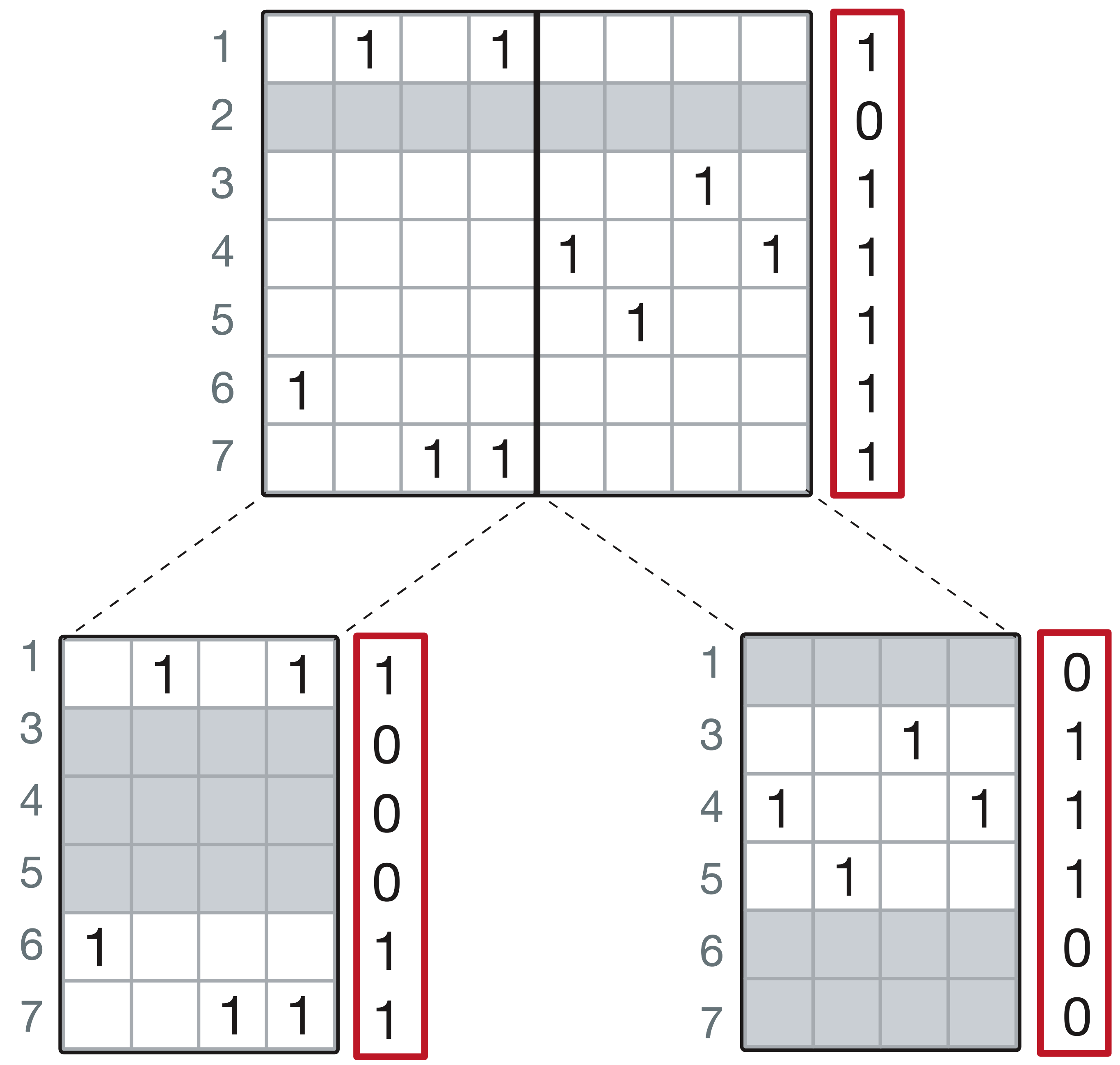
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**



Background

Graph Annotation Representations

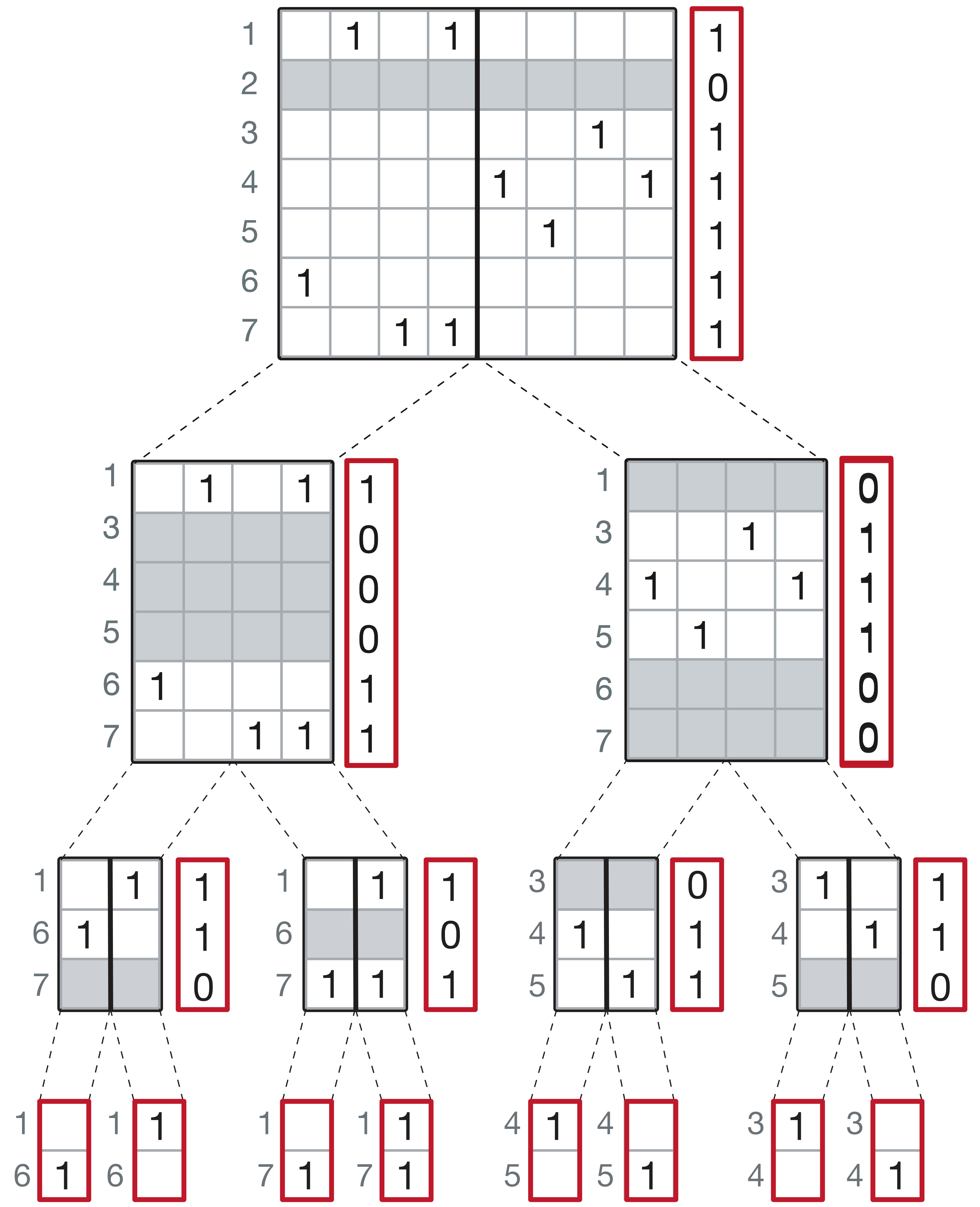
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**



Background

Graph Annotation Representations

1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**



Background

Graph Annotation Representations

1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)

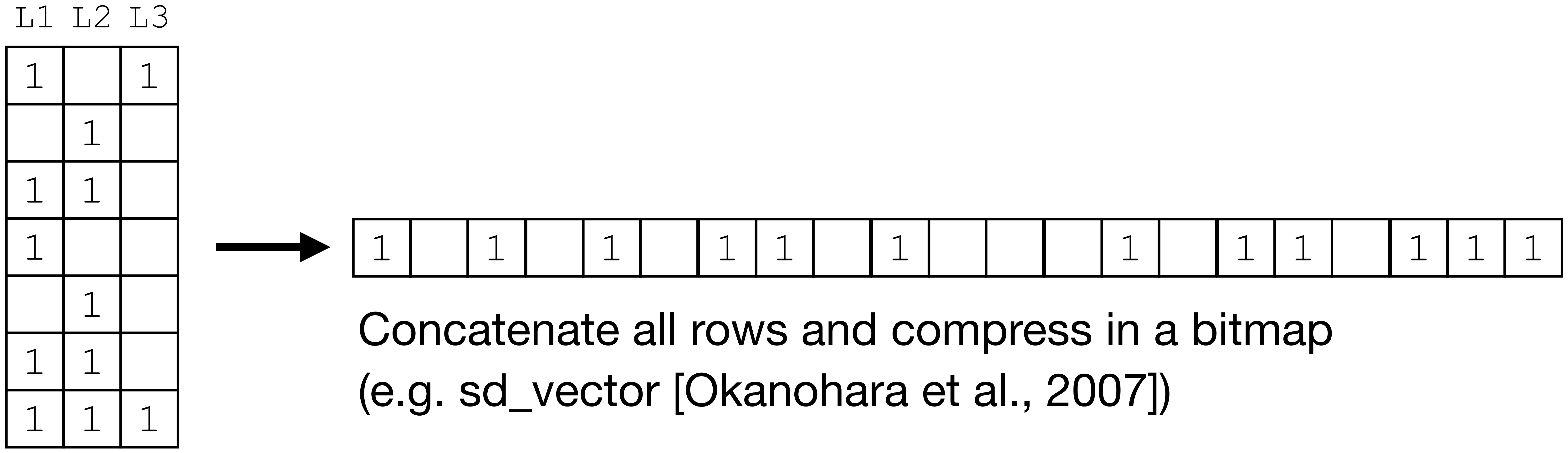
L1	L2	L3
1		1
	1	
1	1	
1		
	1	
1	1	
1	1	1

1		1		1		1	1		1				1		1	1		1	1	1

Background

Graph Annotation Representations

1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)

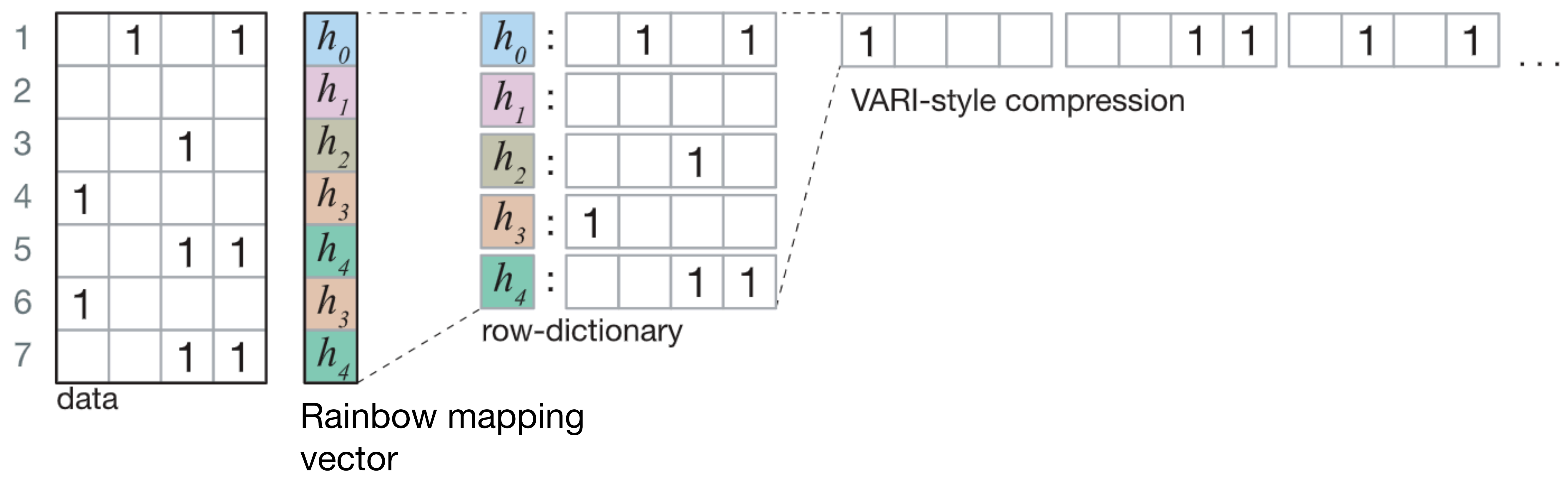


Concatenate all rows and compress in a bitmap
(e.g. sd_vector [Okanohara et al., 2007])

Background

Graph Annotation Representations

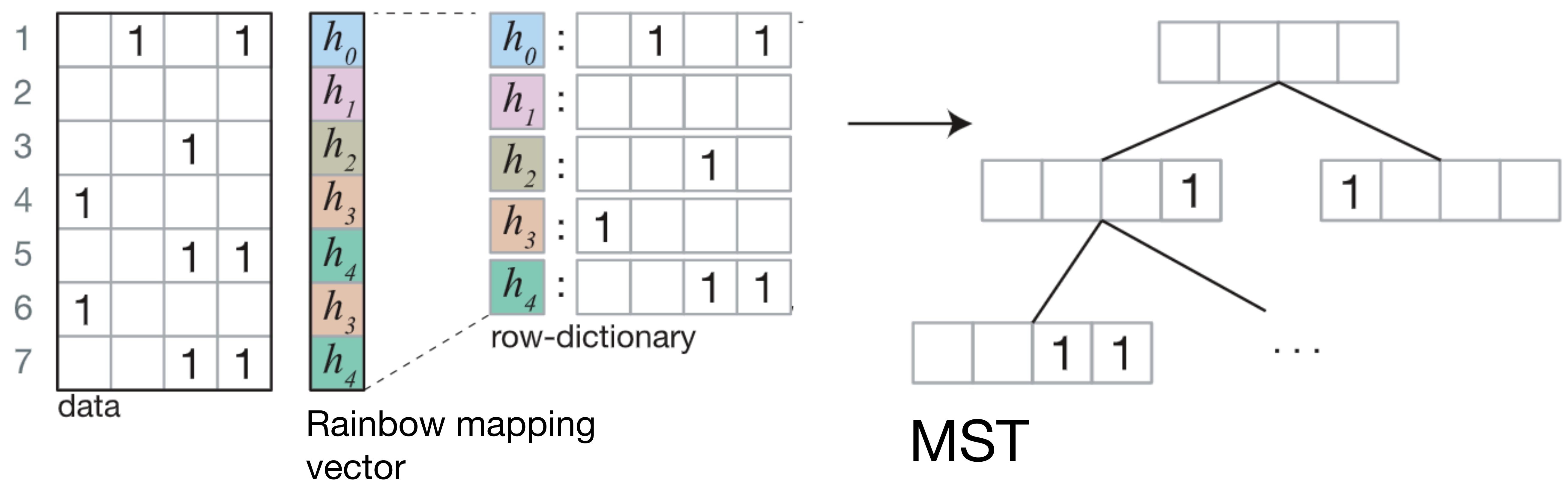
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)
4. Rainbowfish **[Almodaresi et al., 2017]**



Background

Graph Annotation Representations

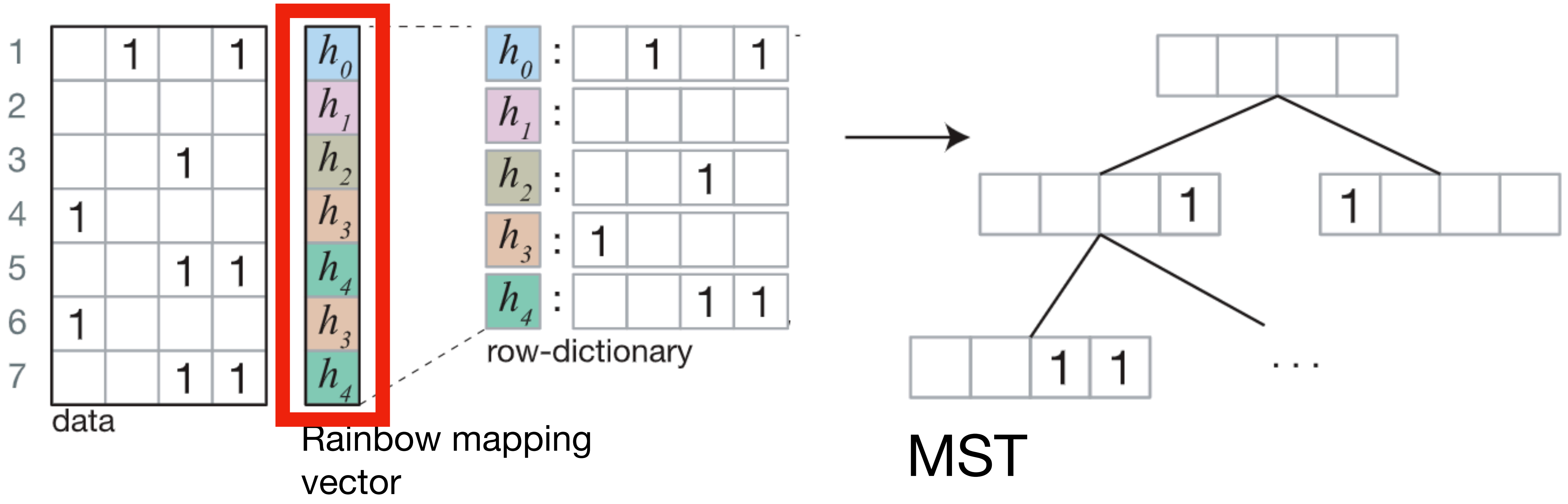
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)
4. Rainbowfish **[Almodaresi et al., 2017]**
5. Mantis-MST **[Almodaresi et al., 2019]**



Background

Graph Annotation Representations

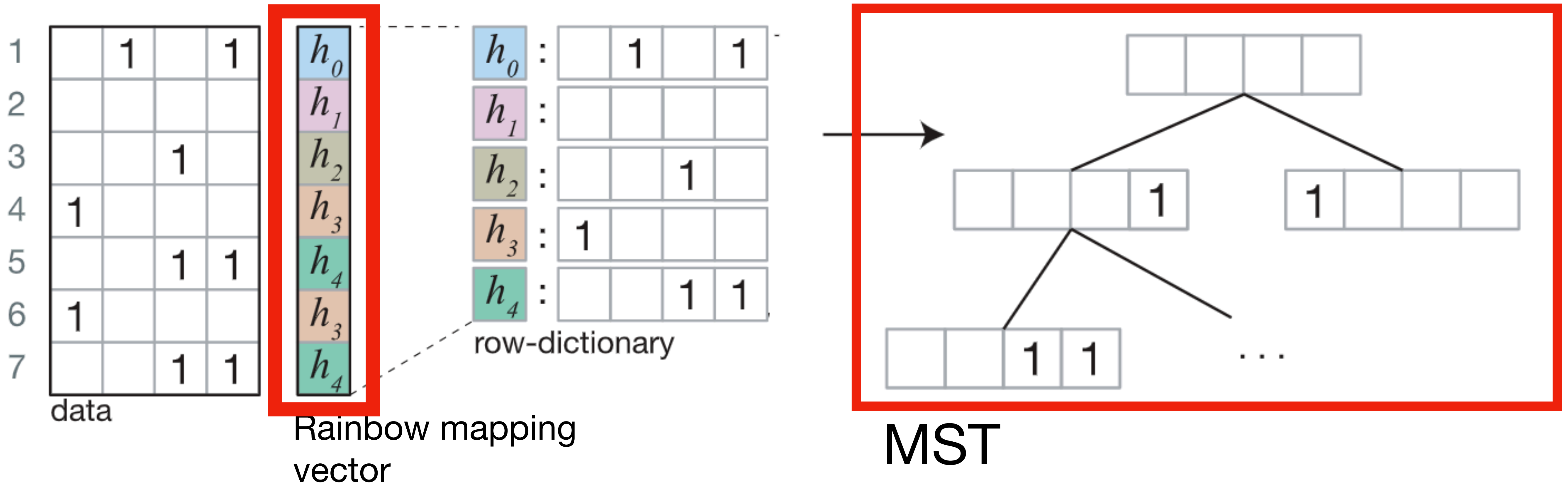
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)
4. Rainbowfish **[Almodaresi et al., 2017]**
5. Mantis-MST **[Almodaresi et al., 2019]**



Background

Graph Annotation Representations

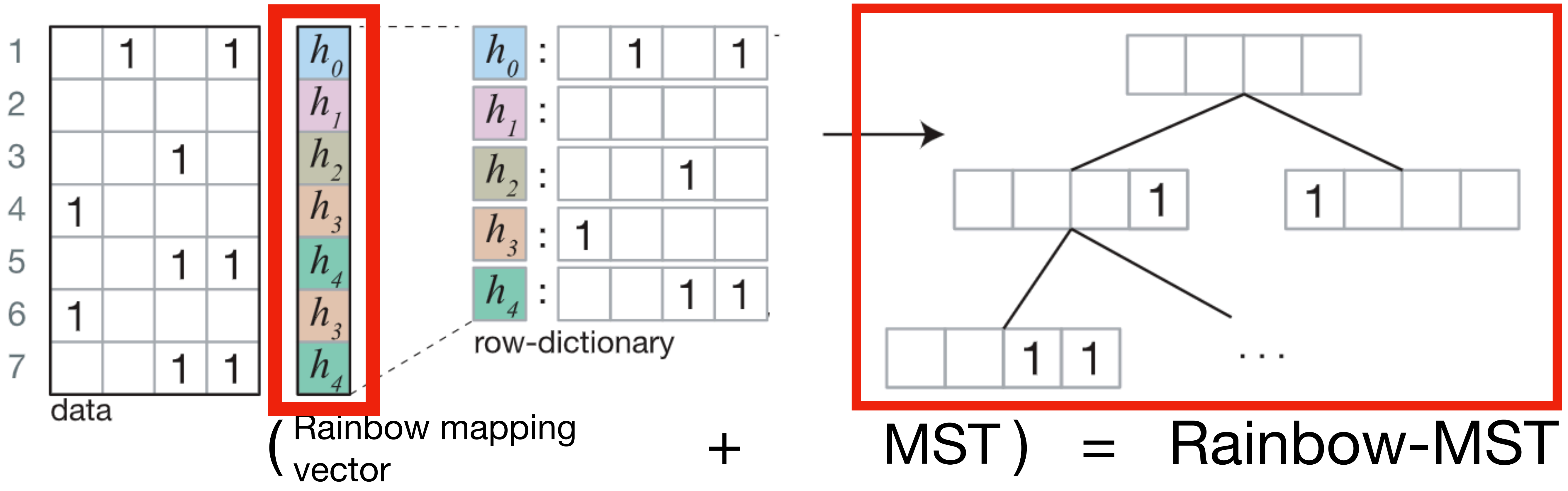
1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)
4. Rainbowfish **[Almodaresi et al., 2017]**
5. Mantis-MST **[Almodaresi et al., 2019]**



Background

Graph Annotation Representations

1. Column-major sparse representation
2. Multi-BRWT **[Karasikov et al., 2019]**
3. RowFlat (employed in VARI **[Muggli et al., 2017]**)
4. Rainbowfish **[Almodaresi et al., 2017]**
5. Mantis-MST **[Almodaresi et al., 2019]**



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

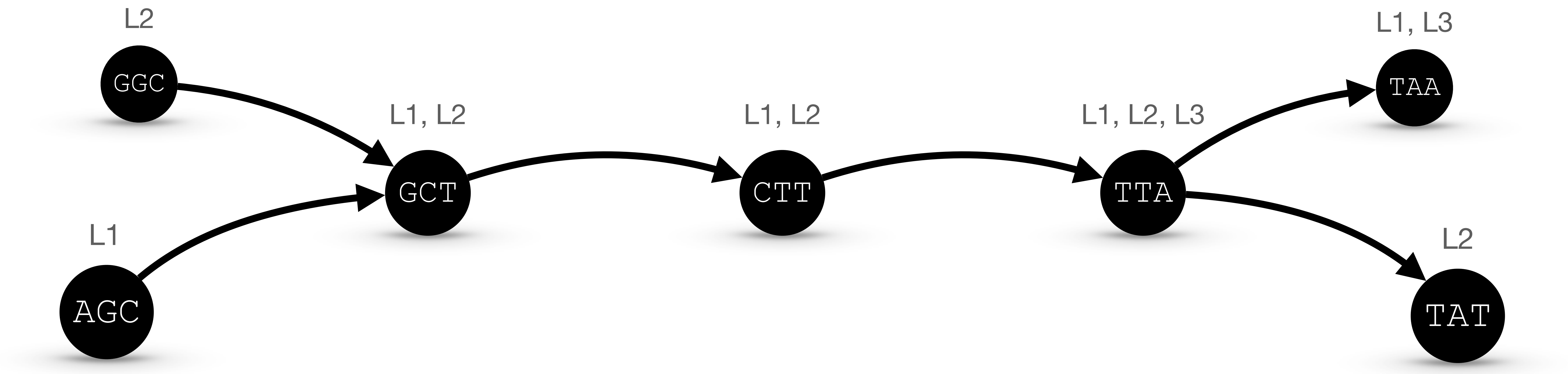
	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

Key idea:

- Store only **diffs**

$$L^{\delta}(v) \;:=\; L(v) \oplus L(v_{\text{succ}})$$

$(\oplus \text{ is XOR})$



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

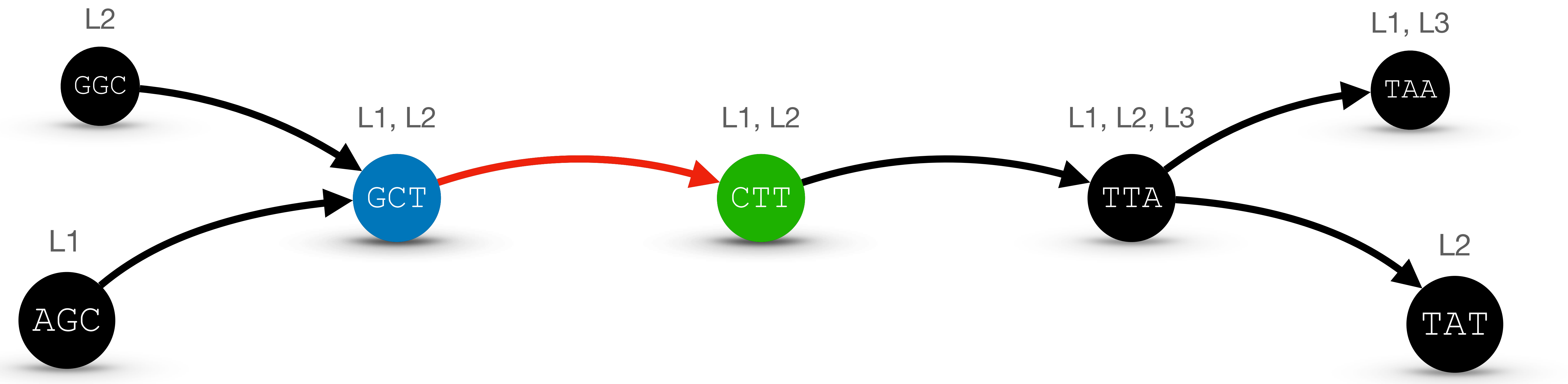
$$\color{red}{L^{\delta}(v)} \ := \ L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3	
TAA	1		1	
TAT		1		
GCT	1	1		v
AGC	1			
GGC		1		
CTT	1	1		v_{succ}
TTA	1	1	1	

→

L1	L2	L3
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

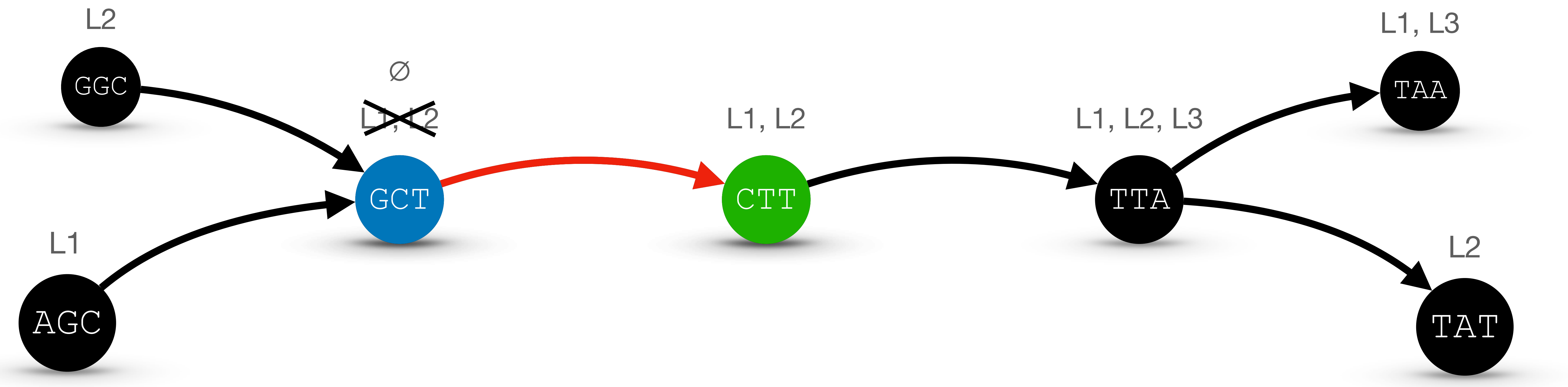
$$\color{red}{L^{\delta}(v)} \;:=\; L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3	
TAA	1		1	
TAT		1		
GCT	1	1		v
AGC	1			
GGC		1		
CTT	1	1		v_{succ}
TTA	1	1	1	

→

L1	L2	L3
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

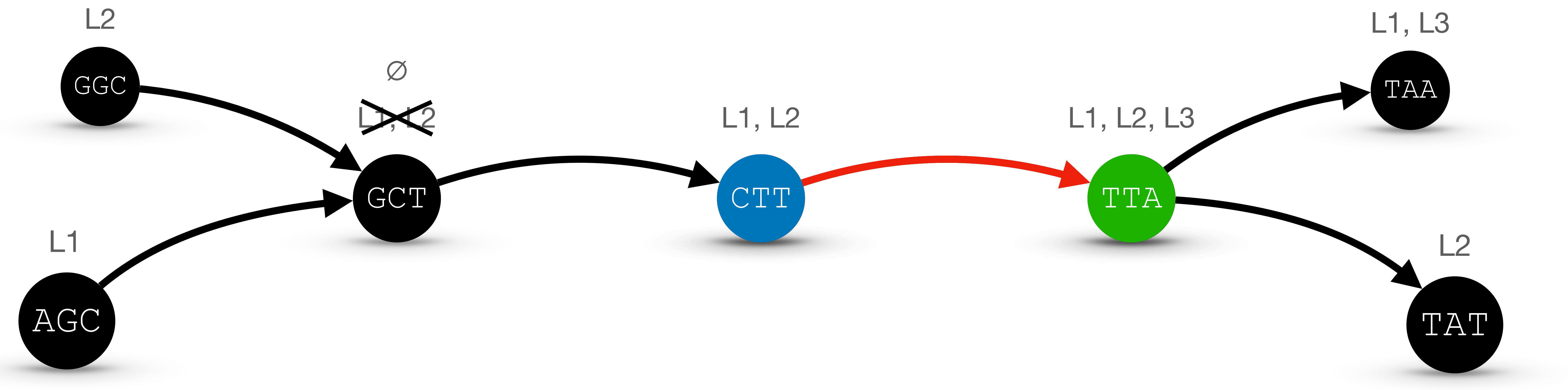
$$L^{\delta}(v) \;:=\; L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3	
TAA	1		1	
TAT		1		
GCT	1	1		
AGC	1			
GGC		1		
CTT	1	1		v
TTA	1	1	1	v_{succ}

$\rightarrow$

L1	L2	L3
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?
?	?	?



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

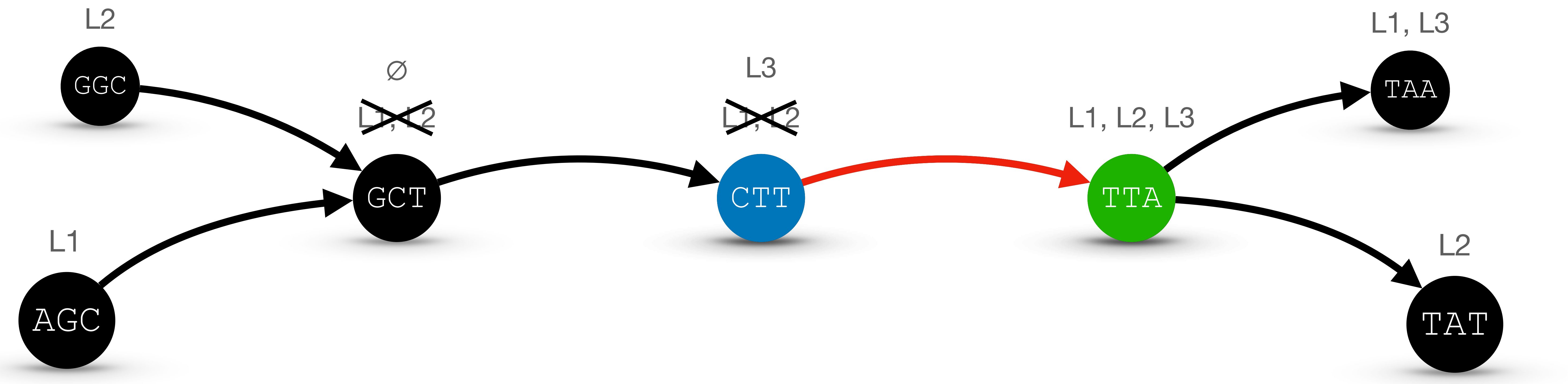
$$\textcolor{red}{L^{\delta}(v)} := L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3	
TAA	1		1	
TAT		1		
GCT	1	1		
AGC	1			
GGC		1		
CTT	1	1		v
TTA	1	1	1	v_{succ}

→

L1	L2	L3
?	?	?
?	?	?
?	?	?
?	?	?
		1
?	?	?



Method

RowDiff Transform

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

$\rightarrow$

L1	L2	L3
?	?	?
?	?	?
?	?	?
?	?	?
		1
?	?	?

Observe:

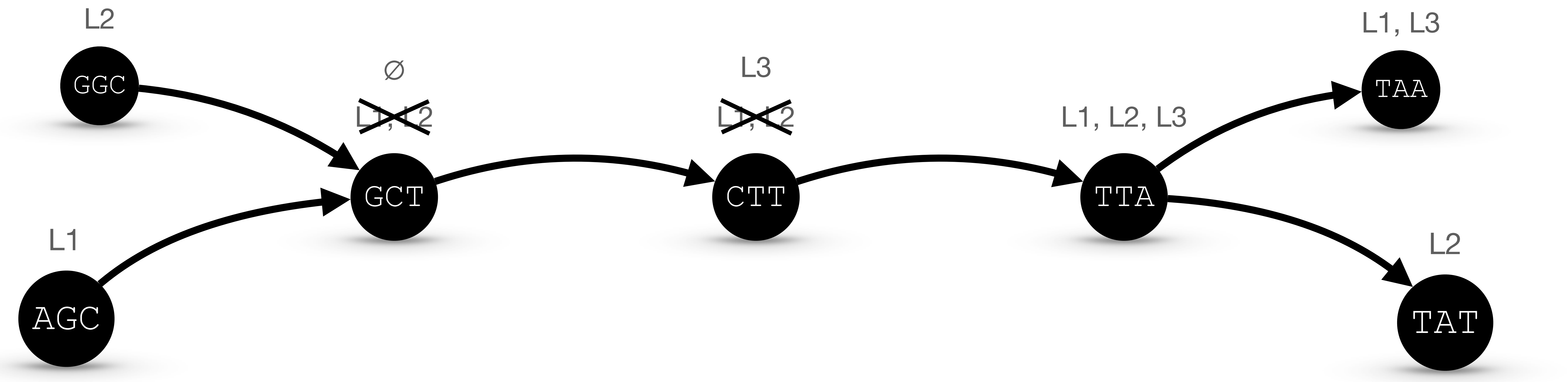
- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

$$L^{\delta}(v) := L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

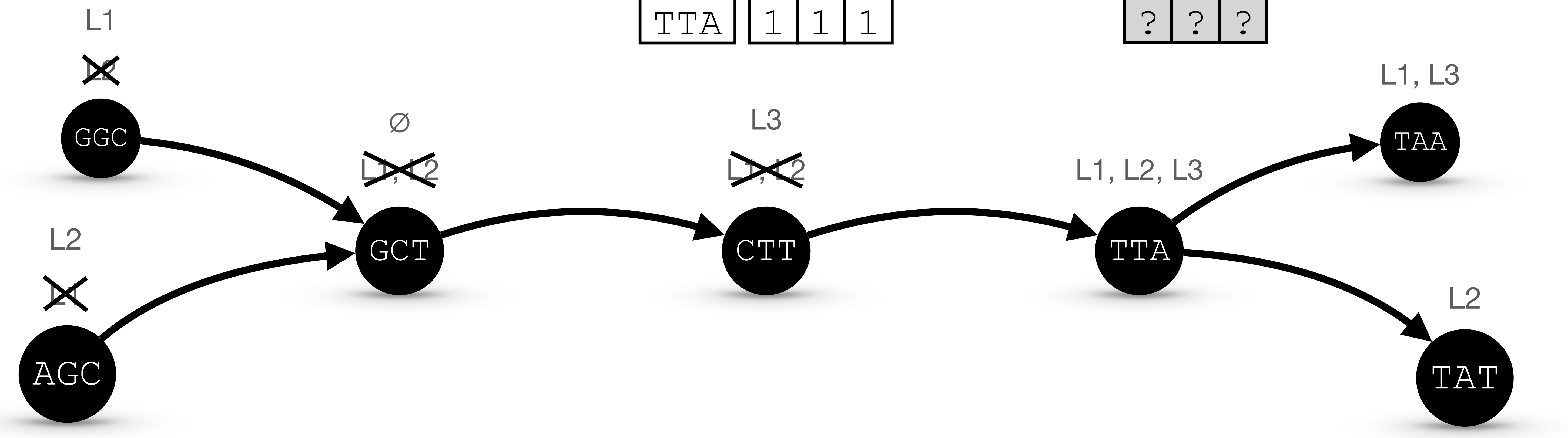
$$\color{red}{L^{\delta}(v)} := L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3
?	?	?
?	?	?
	1	
1		
		1
?	?	?



Method

RowDiff Transform

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3
?	?	?
?	?	?
	1	
1		
		1
?	?	?

Observe:

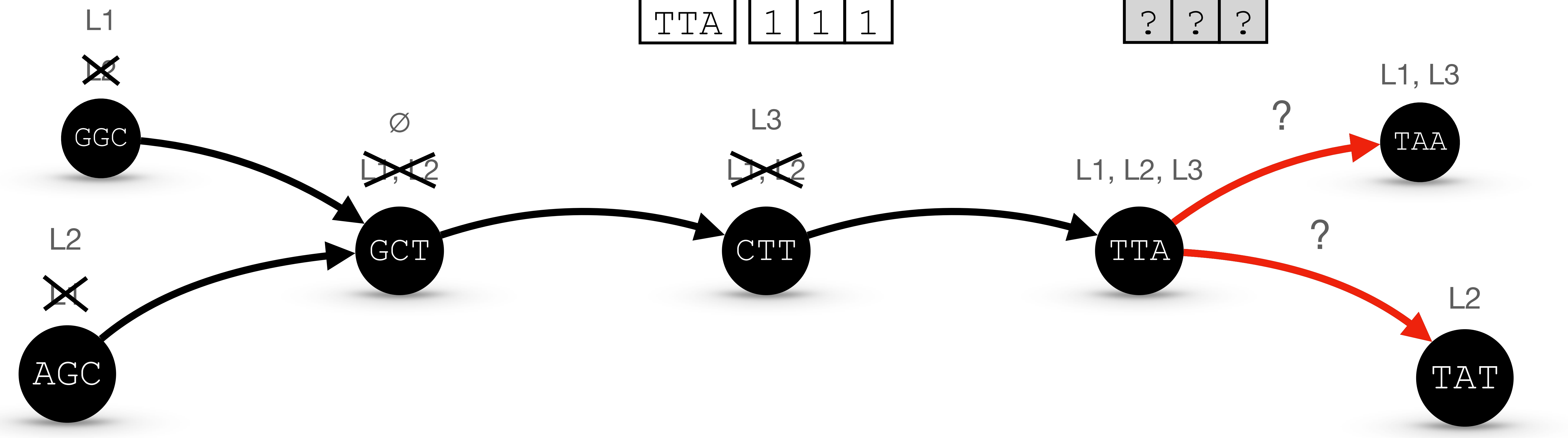
- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

$$L^{\delta}(v) := L(v) \oplus L(v_{\text{succ}})$$

$(\oplus \text{ is XOR})$



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

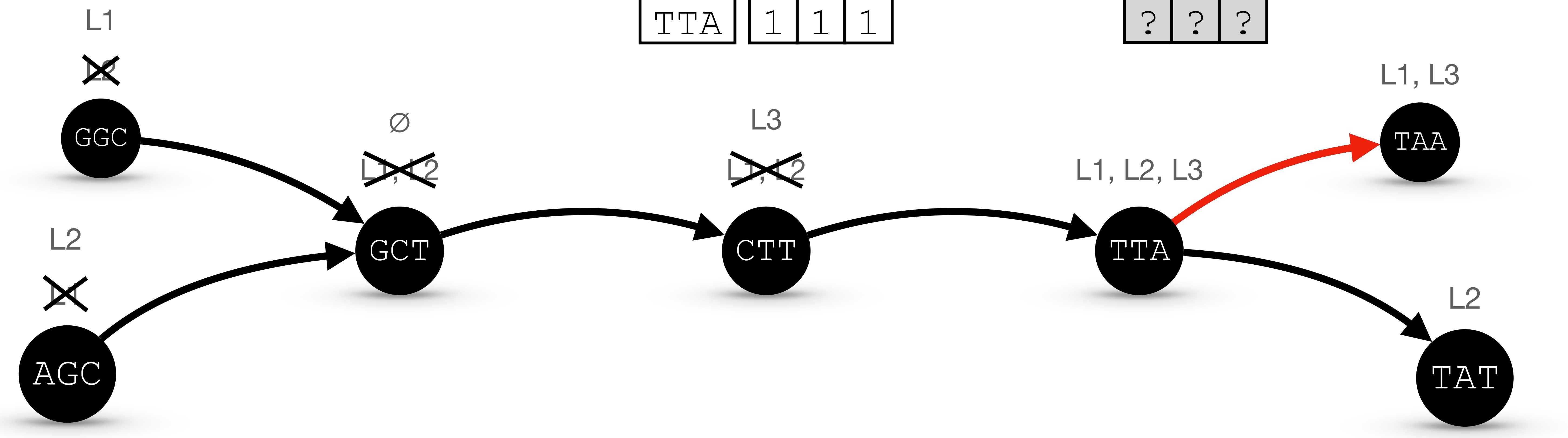
$$\color{red}{L^{\delta}(v)} \;:=\; L(v) \oplus L(v_{\text{succ}})$$

$(\oplus \text{ is XOR})$

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3
?	?	?
?	?	?
	1	
1		
		1
?	?	?



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

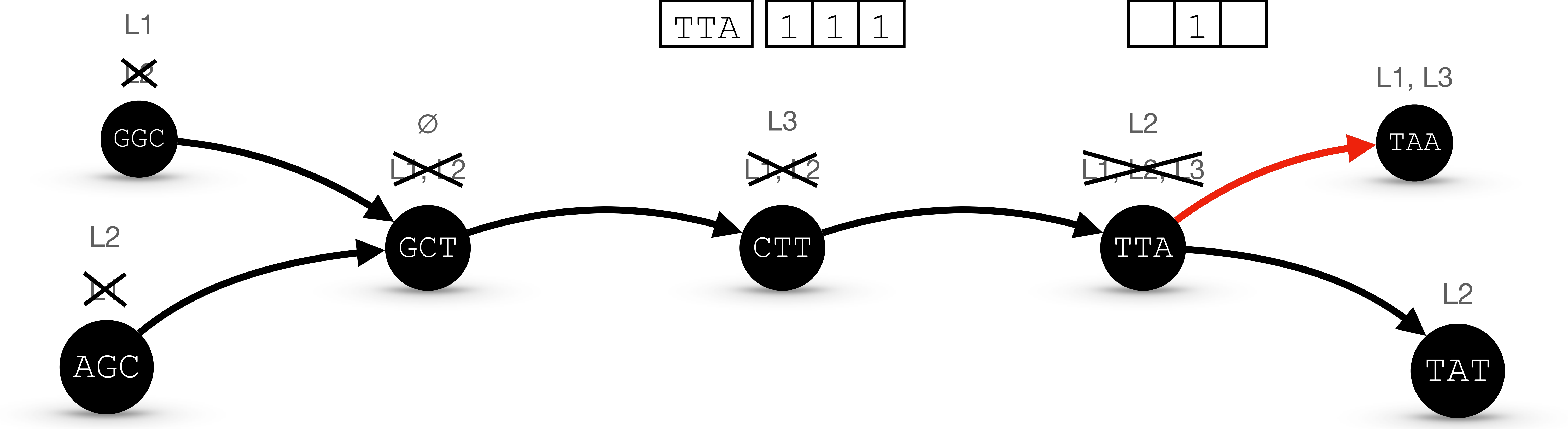
$$L^{\delta}(v) \; := \; L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3
?	?	?
?	?	?
	1	
1		
		1
	1	



Method

RowDiff Transform

Observe:

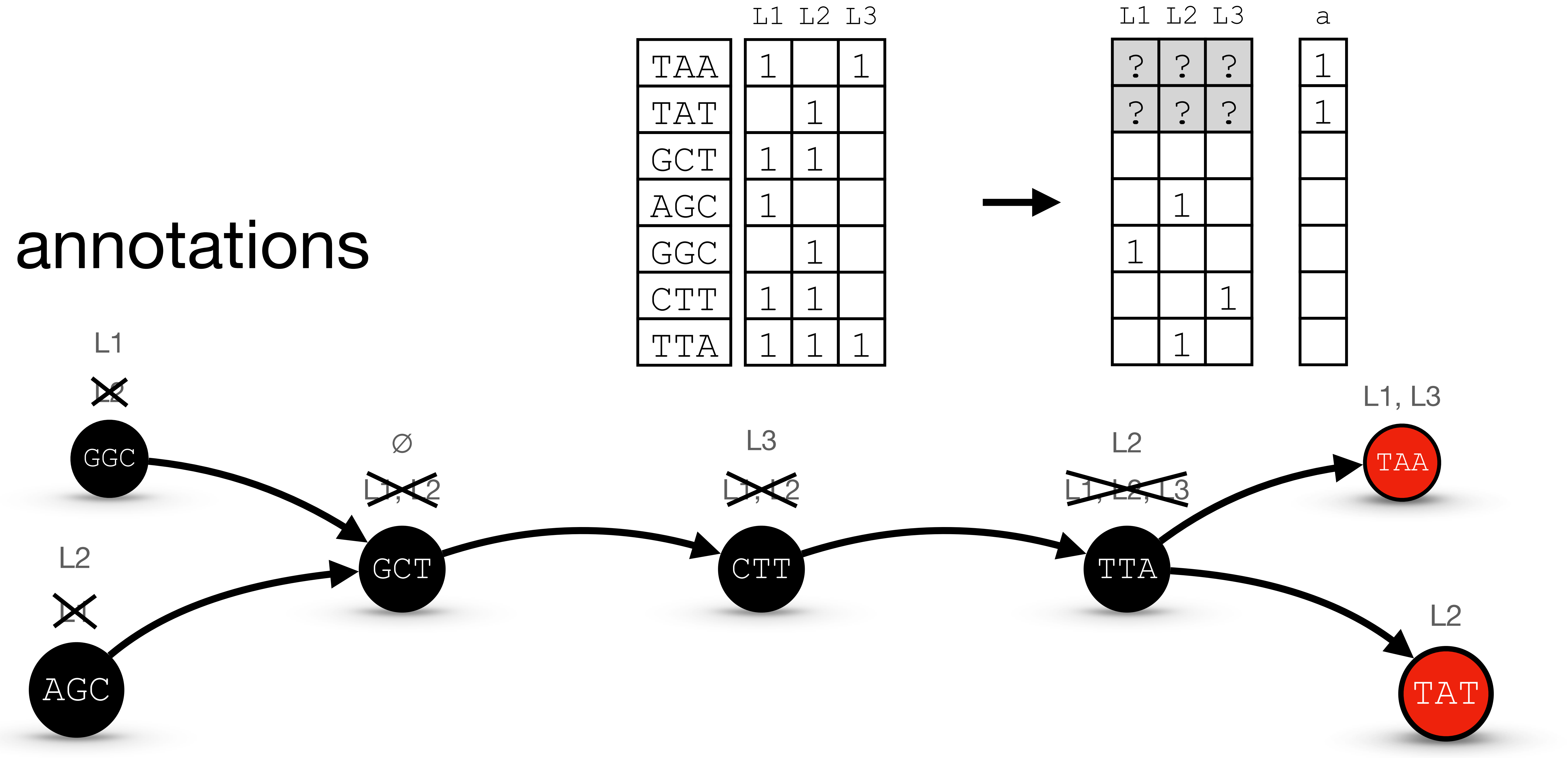
- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

$$\color{red}{L^{\delta}(v)} \; := \; L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$



Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

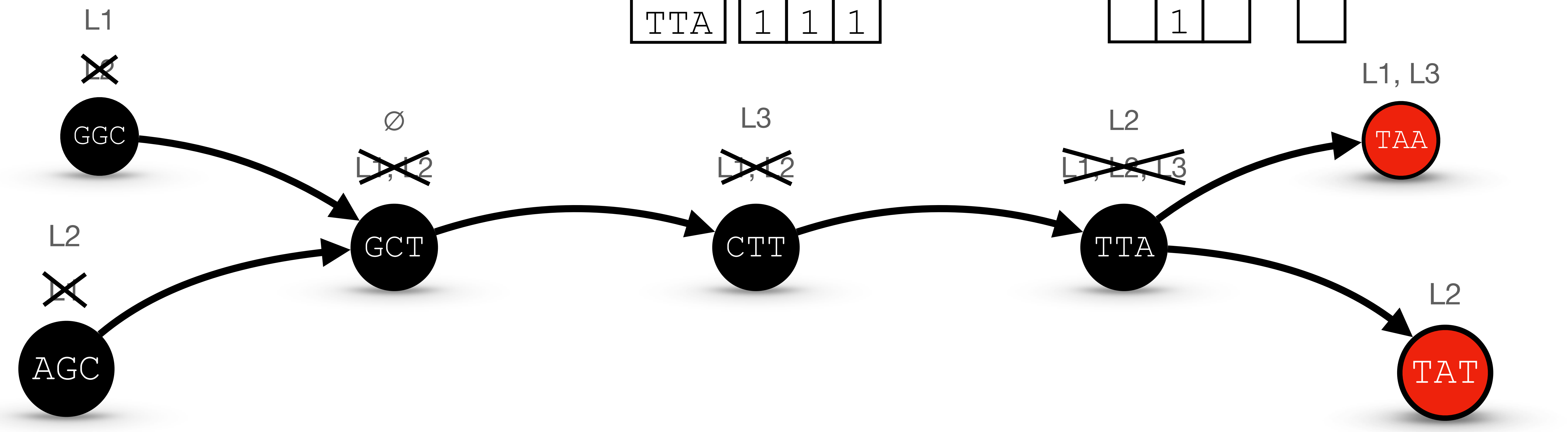
$$\color{red}{L^{\delta}(v)} \;:=\; L(v) \oplus L(v_{\text{succ}})$$

$$(\oplus \text{ is XOR})$$

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3	a
1		1	1
	1		1
	1		
1			
		1	
	1		



RowDiff effectively **transforms** the matrix:

- **makes it sparser**, and thus, more compressible
- can be applied **with any matrix representation**
- the overhead is very small (<1 bit per node)

Method

RowDiff Transform

Observe:

- Adjacent nodes share similar annotations

Key idea:

- Store only **diffs**

$$\color{red}{L^{\delta}(v)} \;:=\; L(v) \oplus L(v_{\text{succ}})$$

- Reconstruct

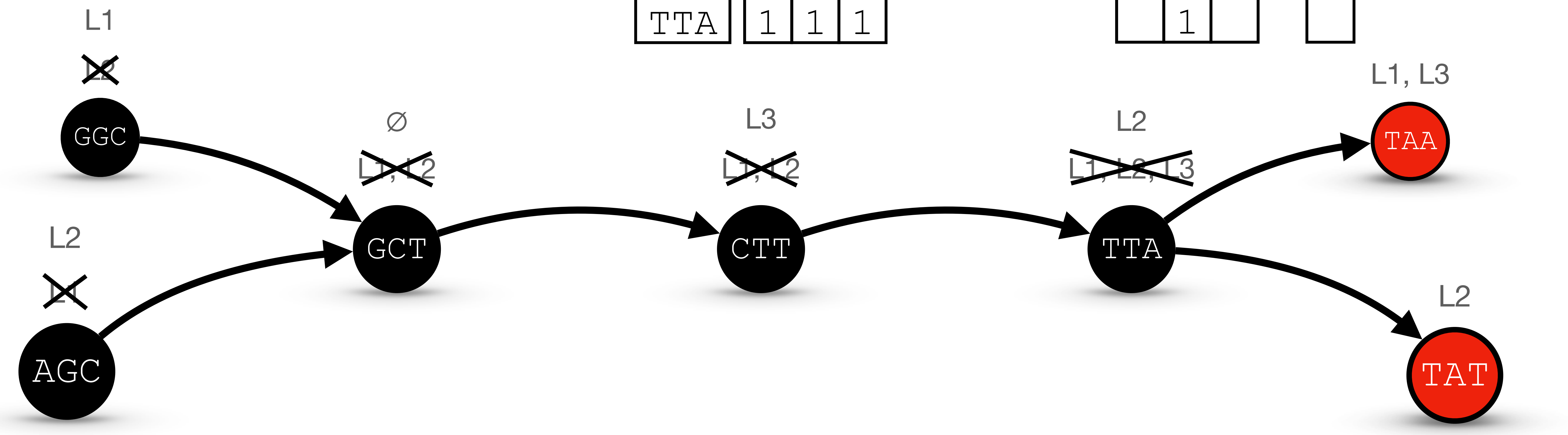
$$L(v) \;=\; L(v_{\text{succ}}) \oplus \color{red}{L^{\delta}(v)}$$

reconstruct recursively

	L1	L2	L3
TAA	1		1
TAT		1	
GCT	1	1	
AGC	1		
GGC		1	
CTT	1	1	
TTA	1	1	1

→

L1	L2	L3	a
1		1	1
	1		1
	1		
1			
		1	
	1		



RowDiff effectively **transforms** the matrix:

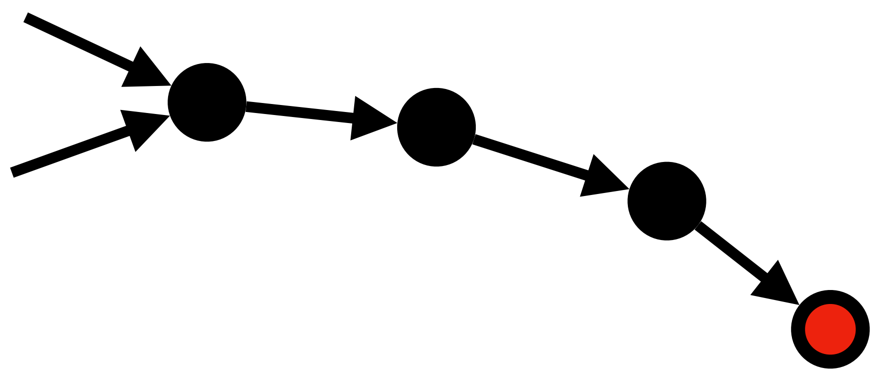
- **makes it sparser**, and thus, more compressible
- can be applied **with any matrix representation**
- the overhead is very small (<1 bit per node)

Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

function ReconstructAnnotation(`i`)
    `row` $\leftarrow A_i^*$
    while $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
        `i` $\leftarrow$ succ(`i`)
        `row` $\leftarrow$ `row` $\oplus A_i^*$
    end while
    return `row`
end function



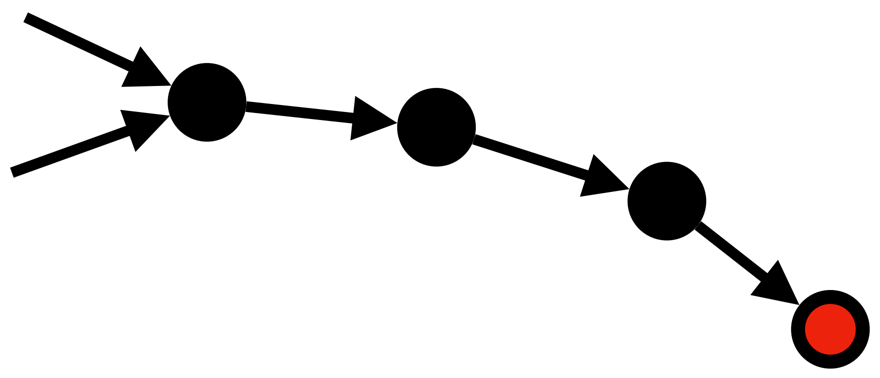
Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

function ReconstructAnnotation(`i`)
    `row` $\leftarrow A_i^*$
    while $\boldsymbol{a}_i = 0$ **do** $\triangleright$ current vertex is not an anchor
        `i` $\leftarrow$ succ(`i`)
        `row` $\leftarrow$ `row` $\oplus A_i^*$
    end while
    return `row`
end function

1. Every *sink node* (with no outgoing edges) must be anchored



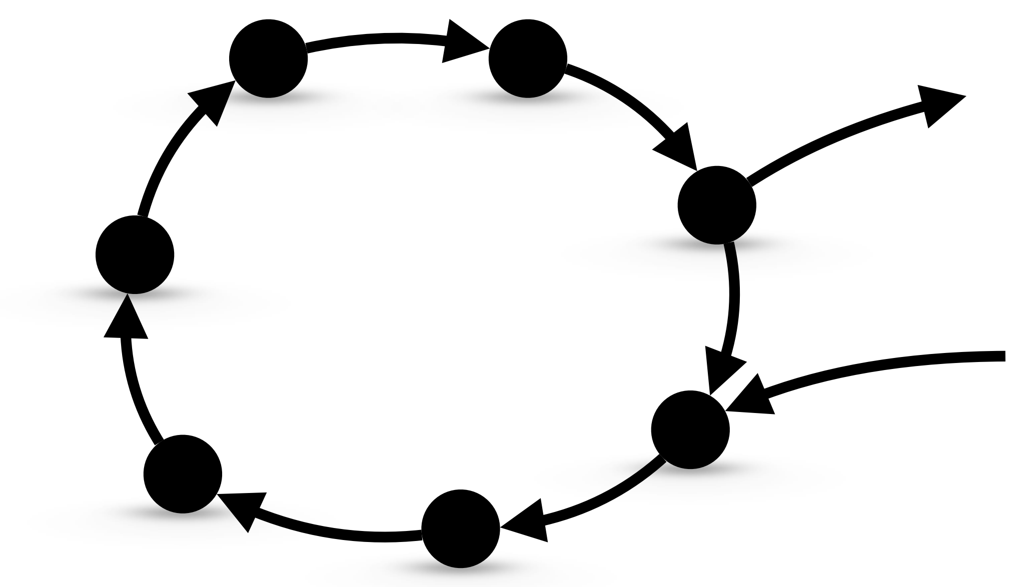
Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

function ReconstructAnnotation(`i`)
  `row` $\leftarrow A_i^*$
  while $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
    `i` $\leftarrow$ succ(`i`)
    `row` $\leftarrow$ `row` $\oplus A_i^*$
  end while
  return `row`
end function

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it



Method

RowDiff: Query

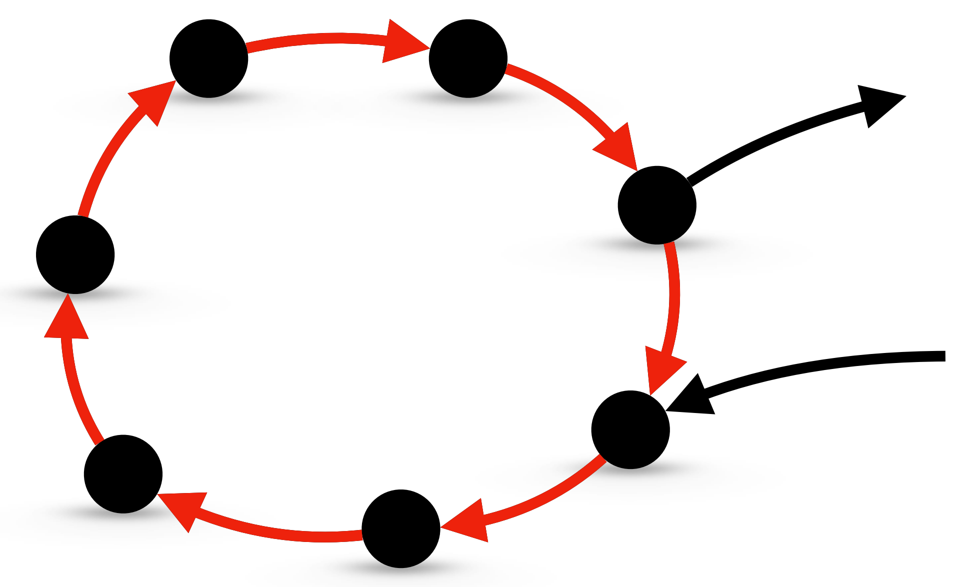
Algorithm 1 Row annotation reconstruction

---

function ReconstructAnnotation($\mathtt{i}$)
    $\mathtt{row} \leftarrow A_i^*$
    while $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
        $\mathtt{i} \leftarrow \mathrm{succ}(\mathtt{i})$
        $\mathtt{row} \leftarrow \mathtt{row} \oplus A_i^*$
    end while
    return $\mathtt{row}$
end function

---

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it



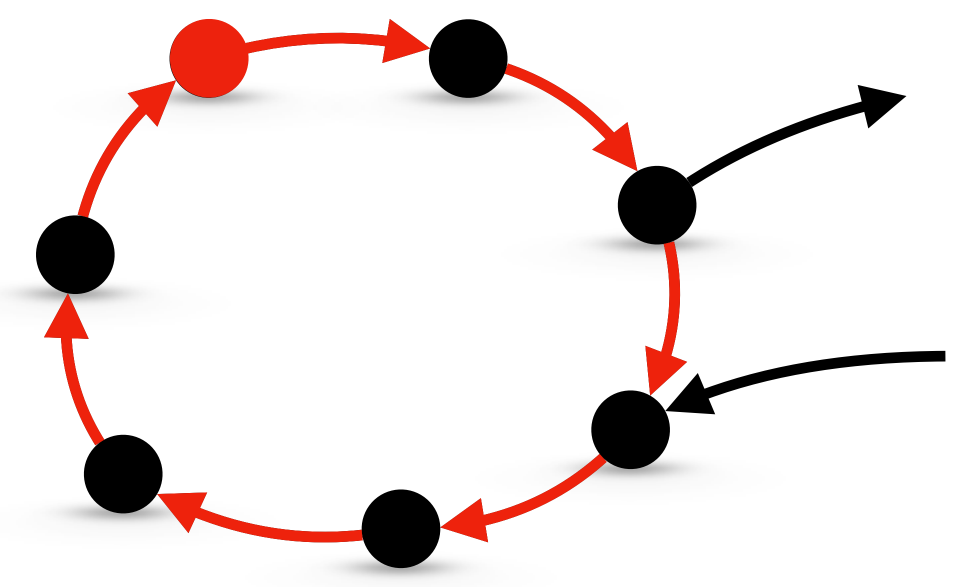
Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

1: **function** ReconstructAnnotation(`i`)
2:     `row` $\leftarrow A_i^*$
3:     **while** $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
4:         `i` $\leftarrow$ succ(`i`)
5:         `row` $\leftarrow$ `row` $\oplus A_i^*$
6:     **end while**
7:     **return** `row`
8: **end function**

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it



Method

RowDiff: Query

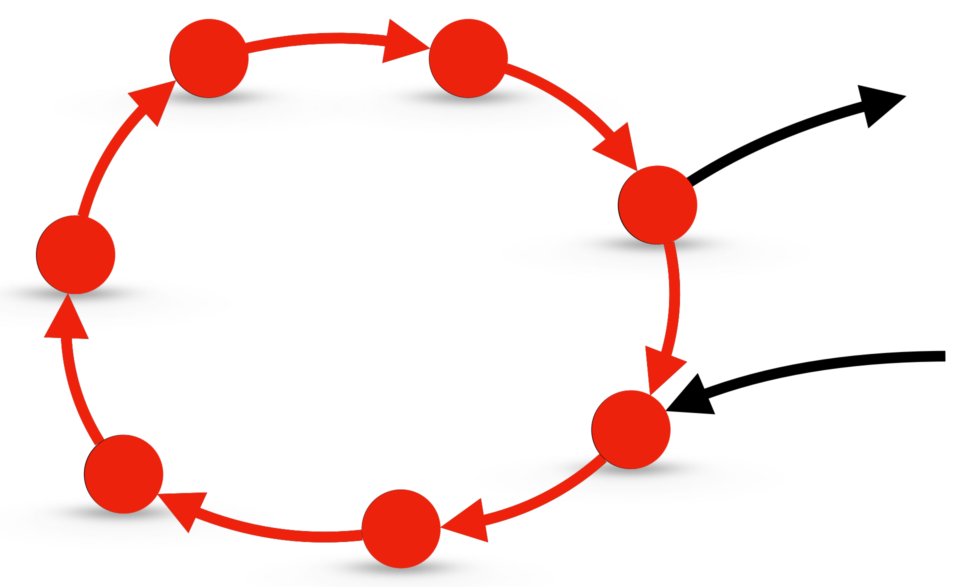
Algorithm 1 Row annotation reconstruction

---

function ReconstructAnnotation($\mathtt{i}$)
  $\mathtt{row} \leftarrow A^*_i$
  while $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
    $\mathtt{i} \leftarrow \mathrm{succ}(\mathtt{i})$
    $\mathtt{row} \leftarrow \mathtt{row} \oplus A^*_i$
  end while
  return $\mathtt{row}$
end function

---

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it



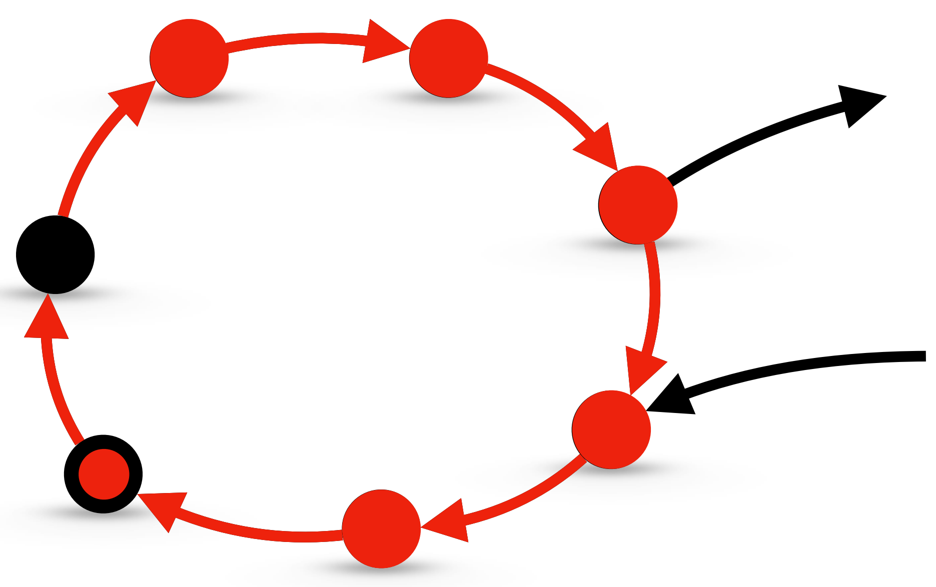
Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

function ReconstructAnnotation($\mathtt{i}$)
  $\mathtt{row} \leftarrow A_i^*$
  while $\boldsymbol{a}_i = 0$ **do** ▷ current vertex is not an anchor
    $\mathtt{i} \leftarrow \mathrm{succ}(\mathtt{i})$
    $\mathtt{row} \leftarrow \mathtt{row} \oplus A_i^*$
  end while
  return $\mathtt{row}$
end function

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it



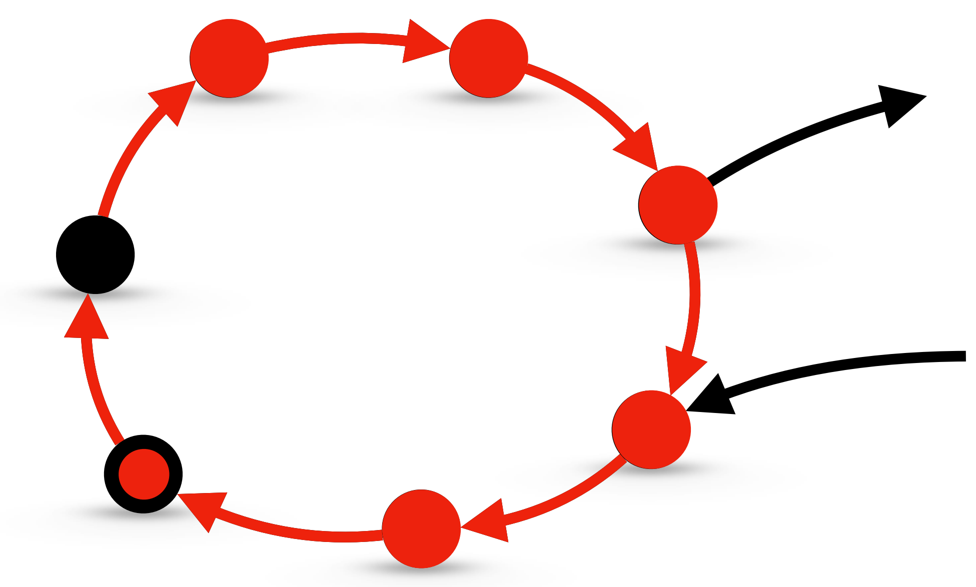
Method

RowDiff: Query

Algorithm 1 Row annotation reconstruction

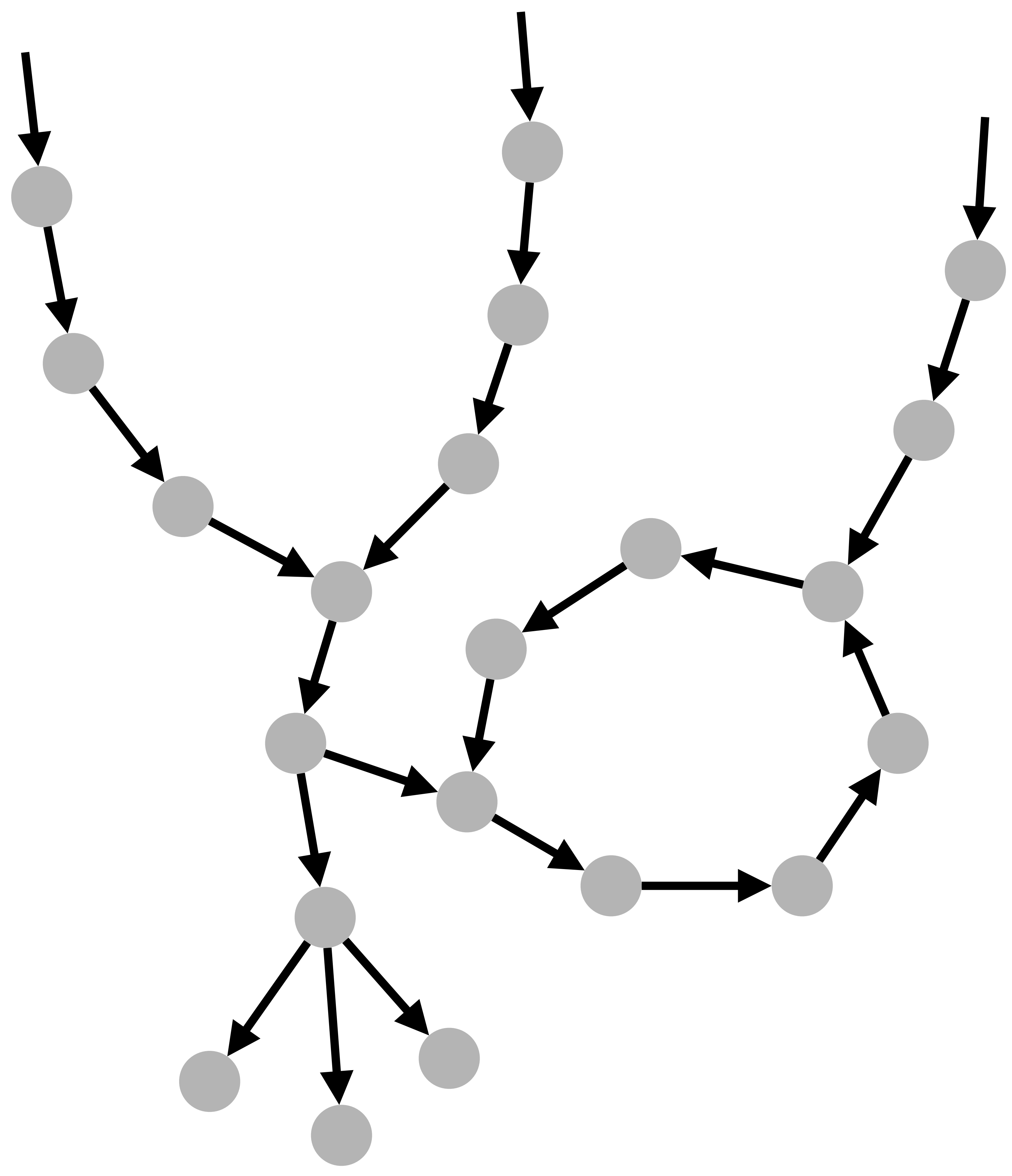
```
function ReconstructAnnotation(i)
    row ← A*_i
    while a_i = 0 do                ▷ current vertex is not an anchor
        i ← succ(i)
        row ← row ⊕ A*_i
    end while
    return row
end function
```

1. Every *sink node* (with no outgoing edges) must be anchored
2. Every *row-diff cycle* must have at least one anchor node in it
3. Length of each row-diff path is bounded by a constant ~M (to ensure a constant query time complexity)



Method

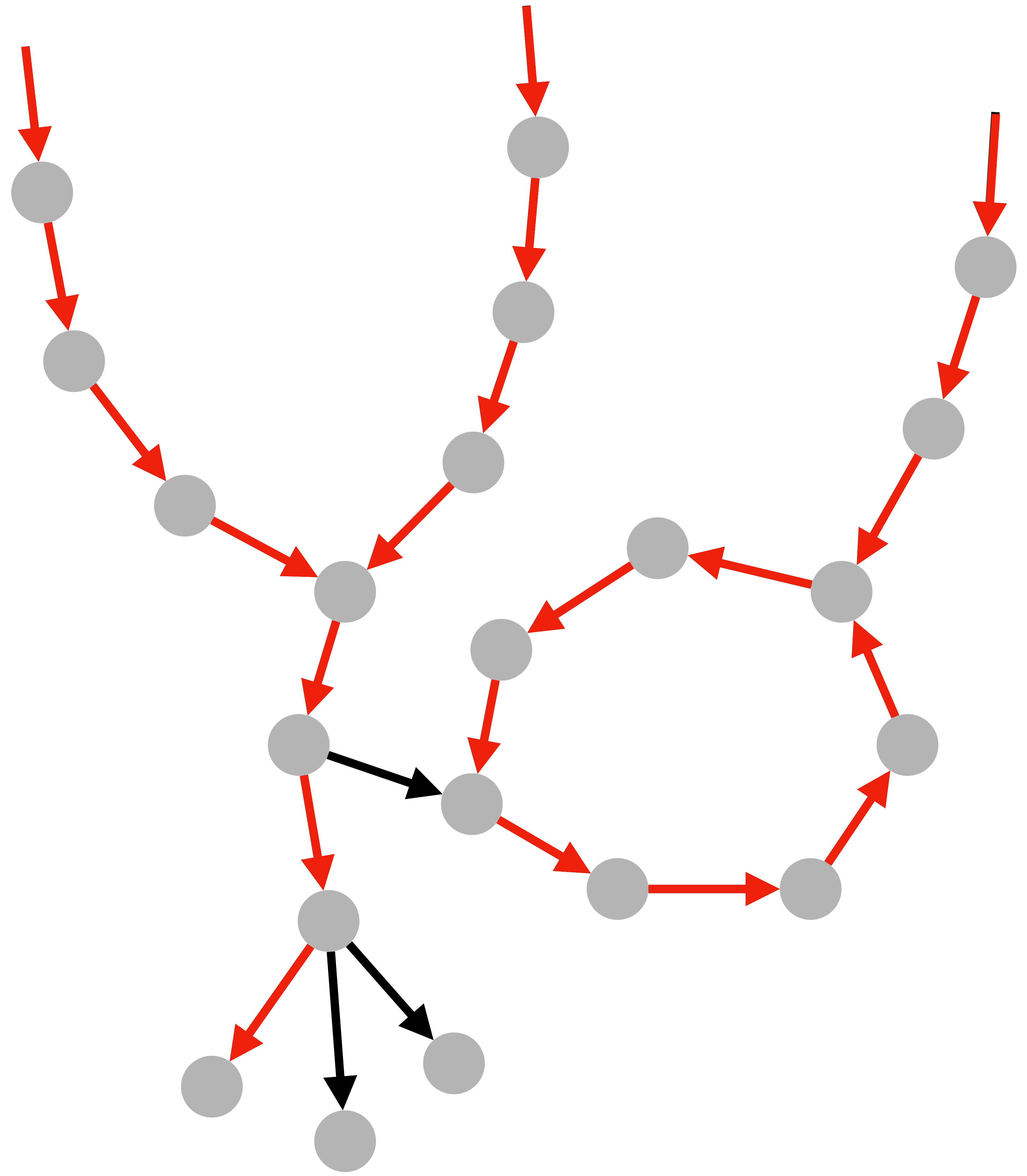
RowDiff: Anchor Assignment



Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

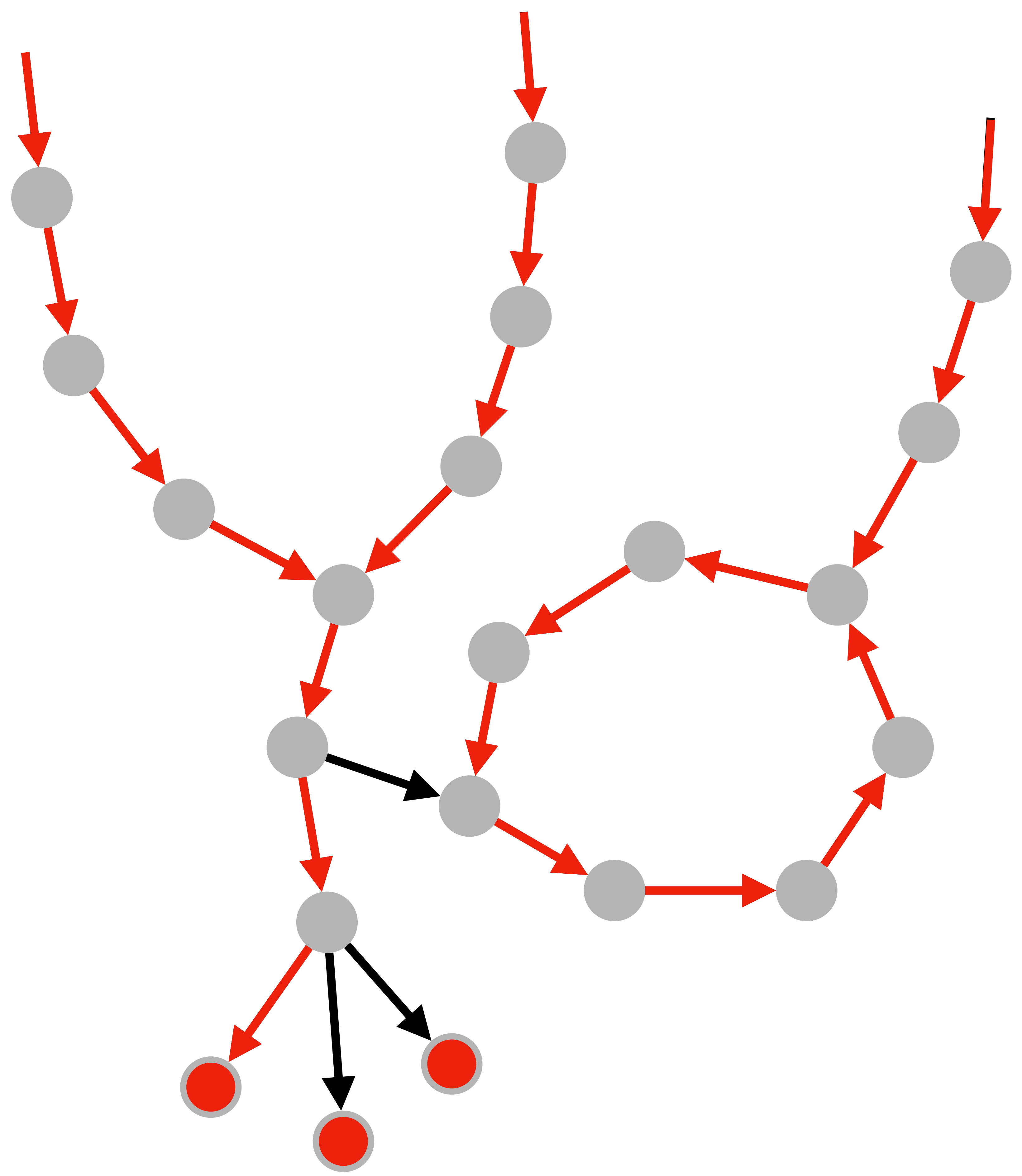


Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**



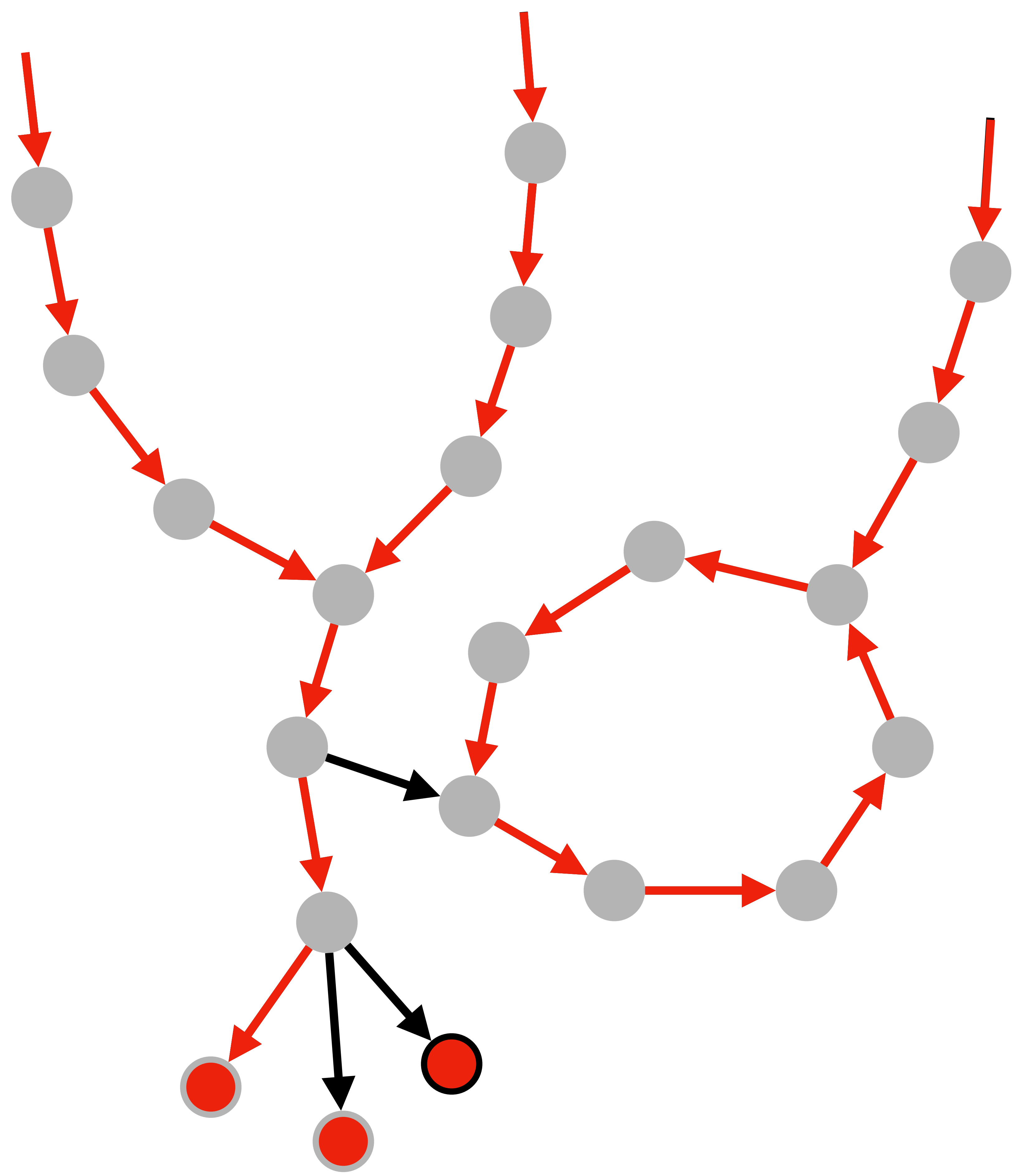
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



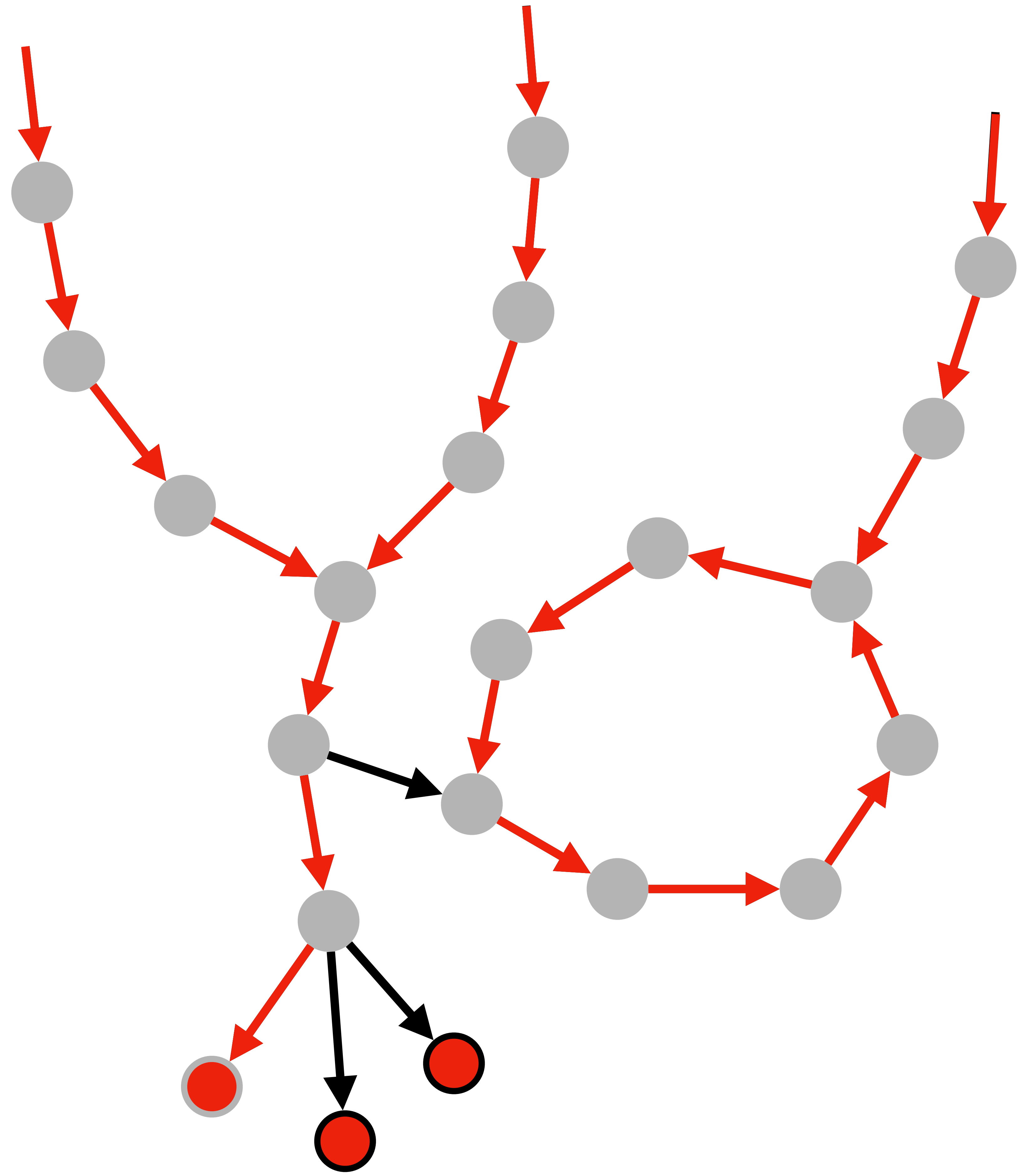
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

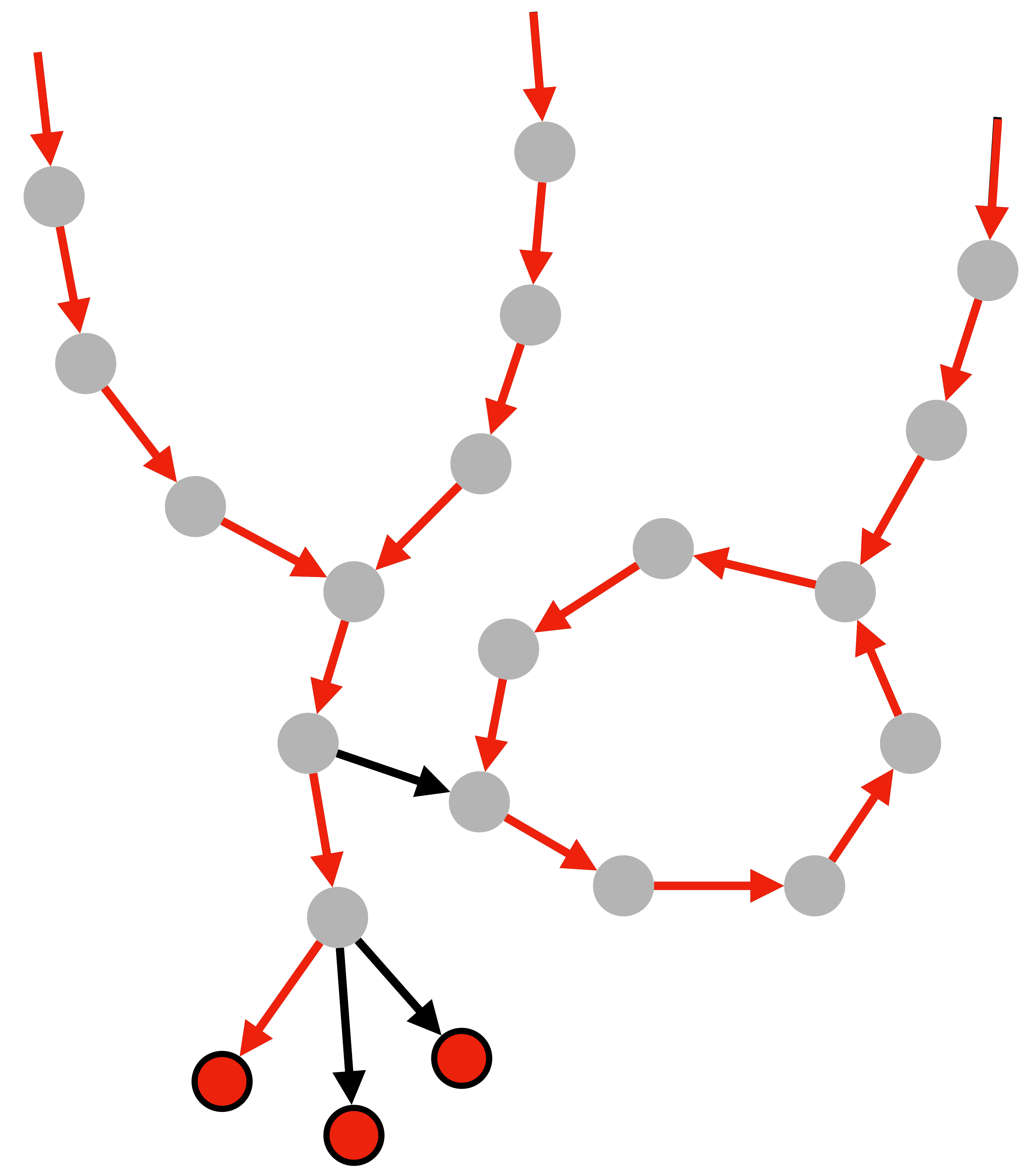
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



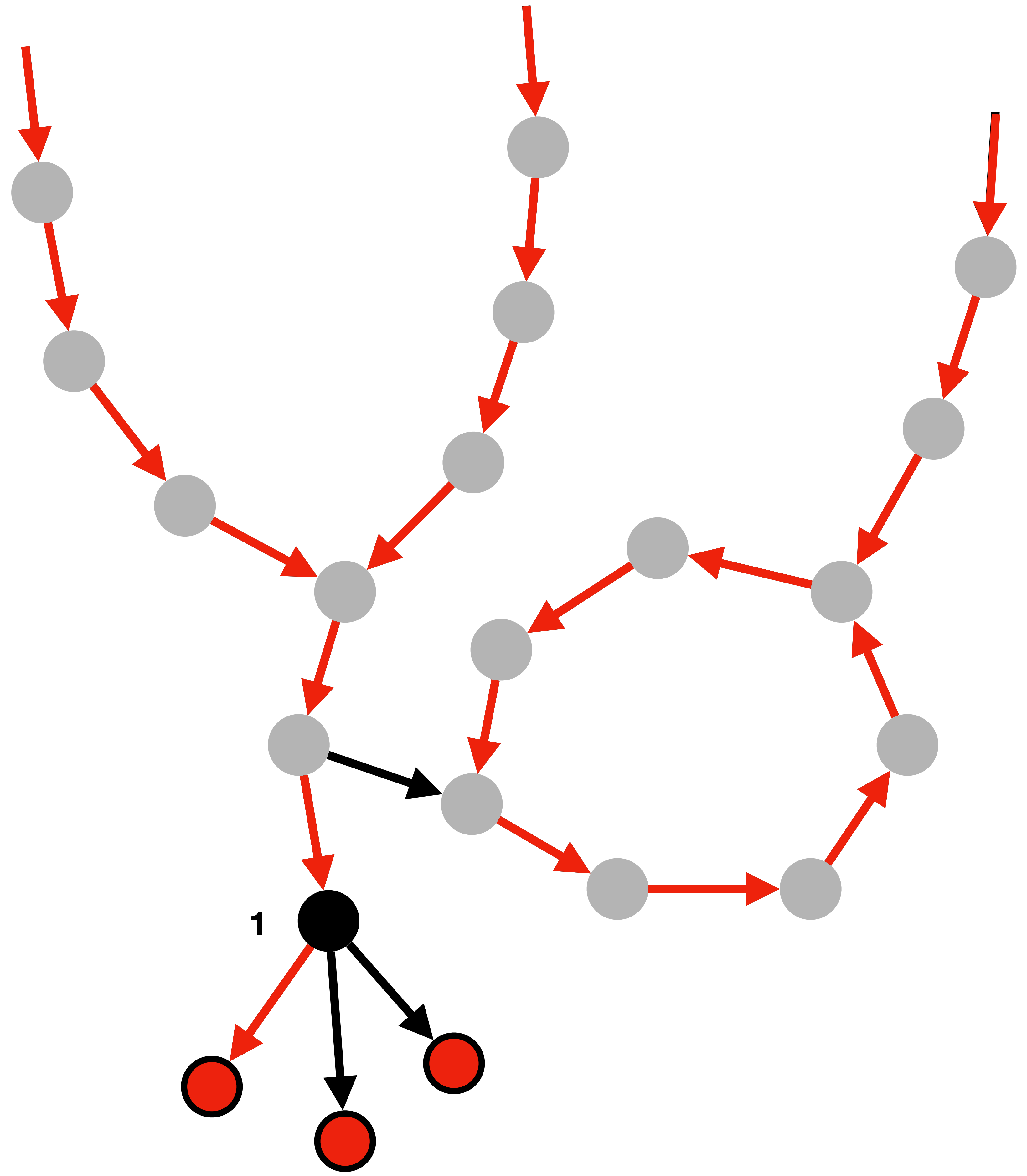
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



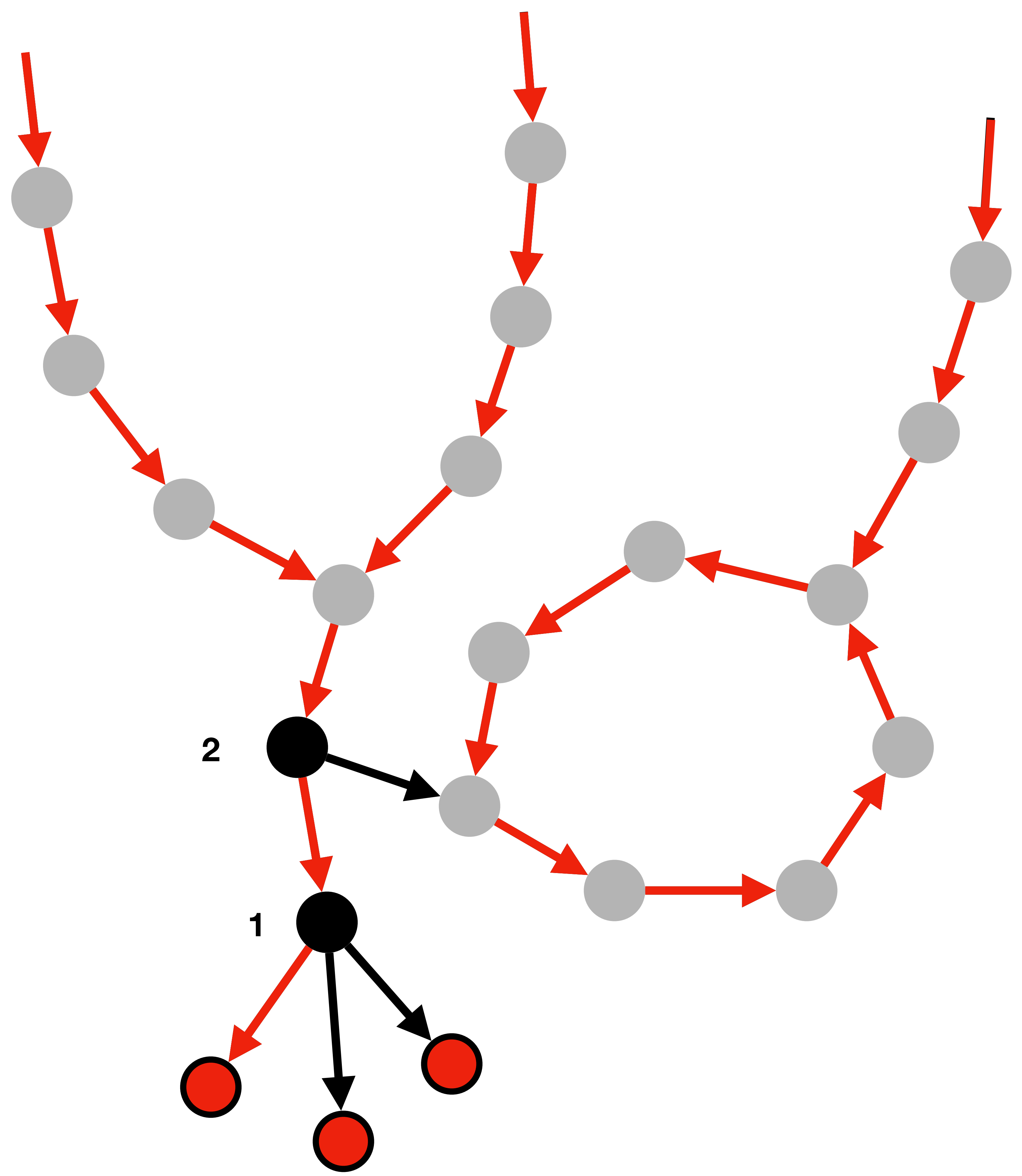
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



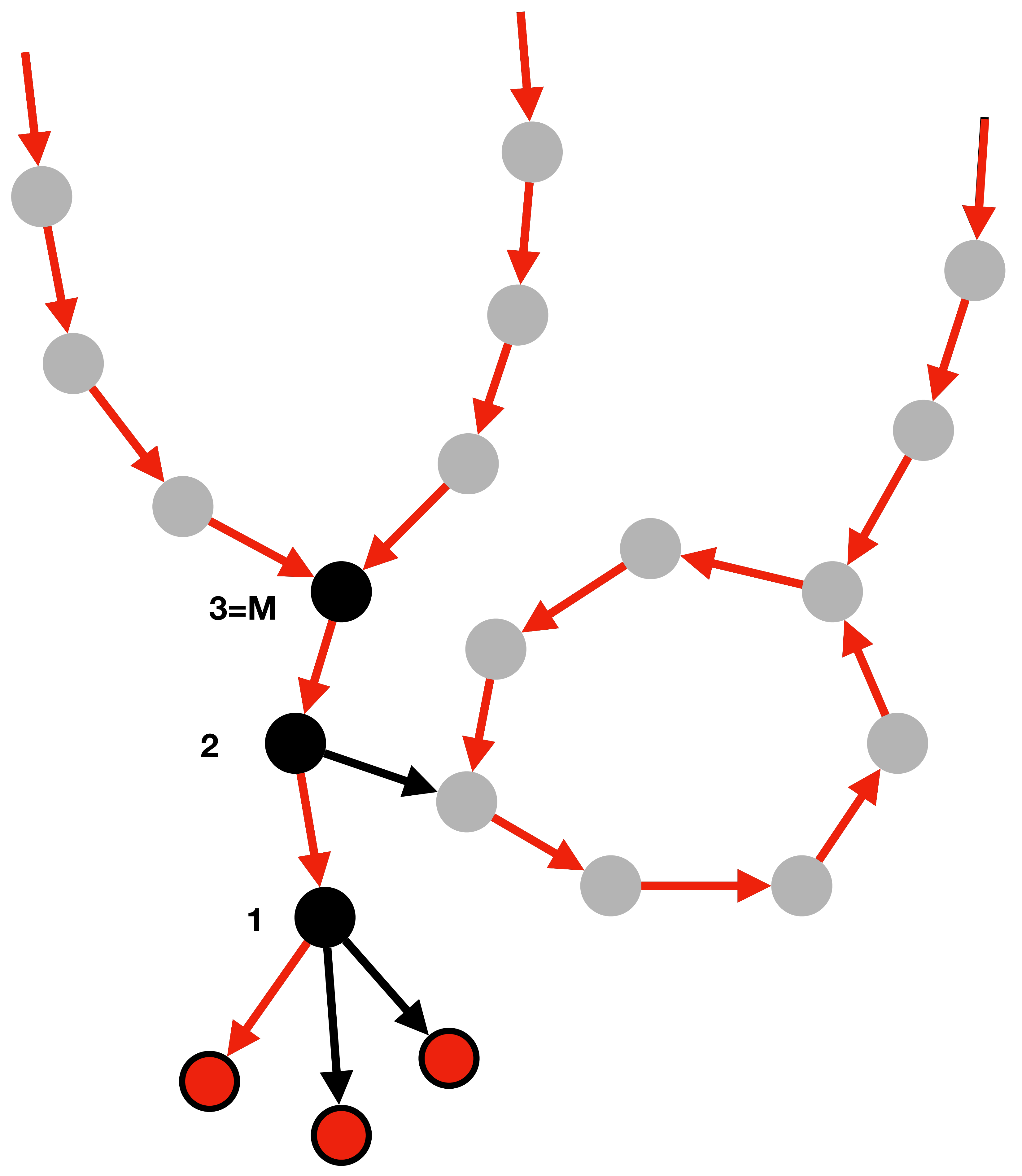
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



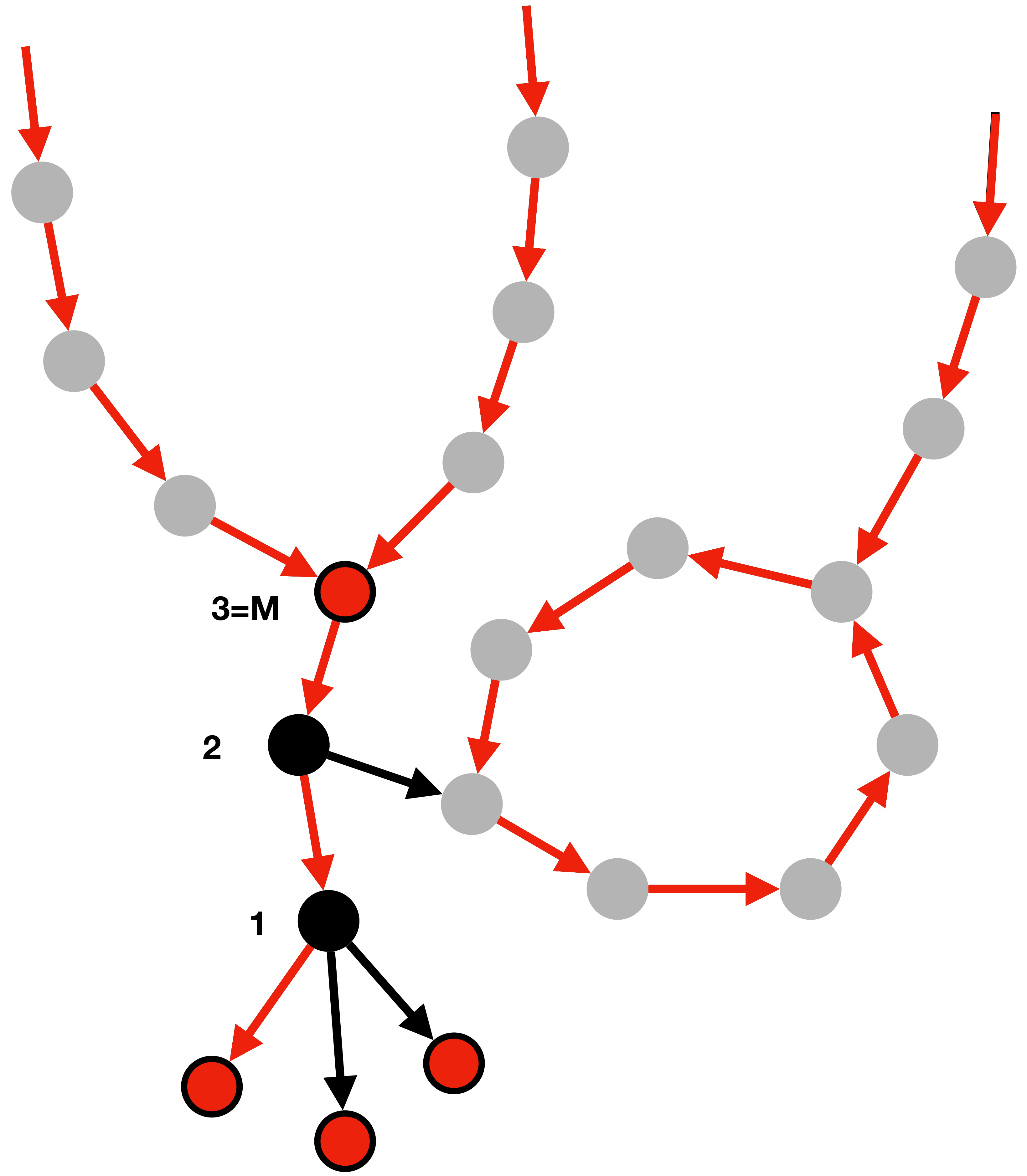
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



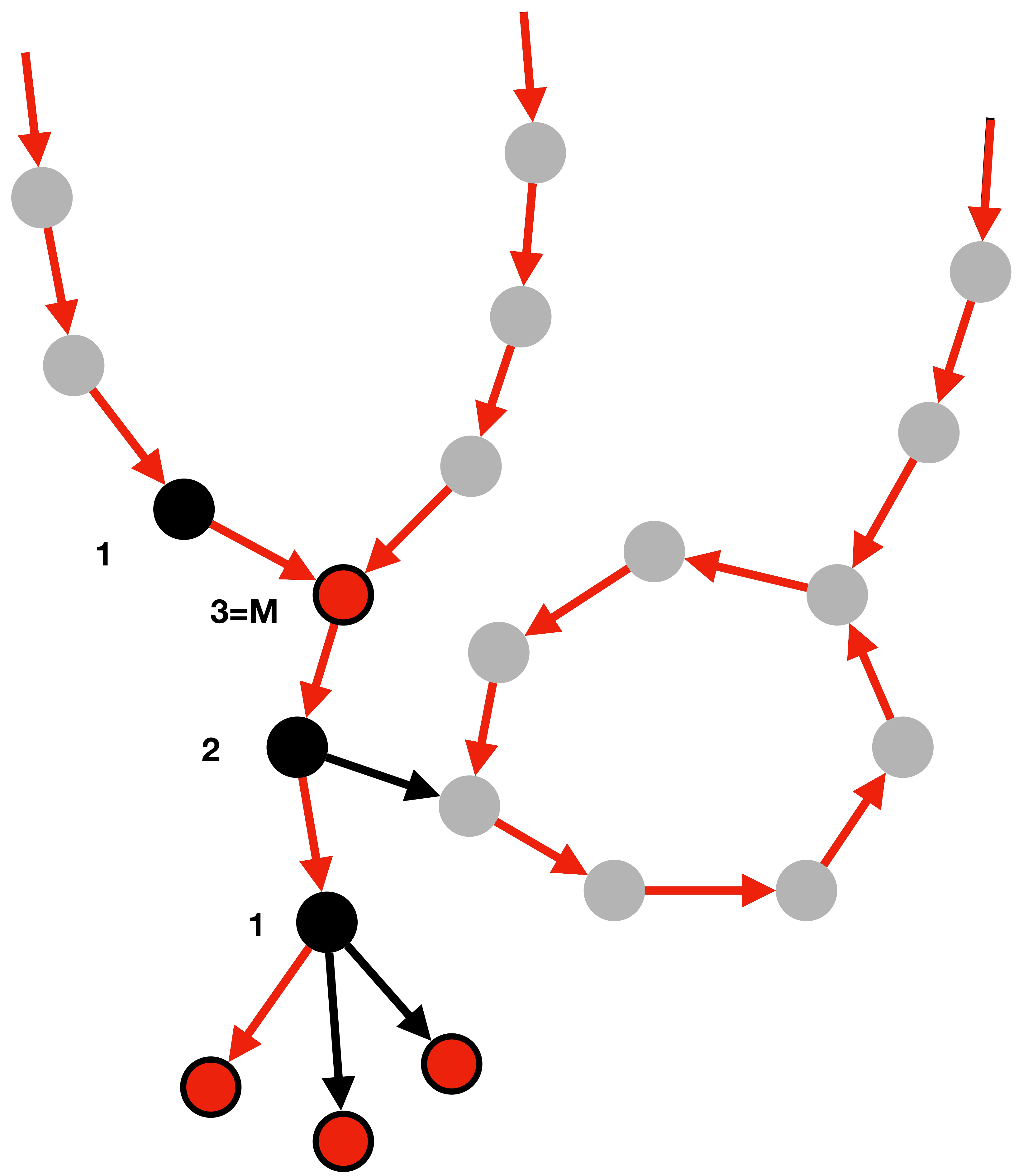
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

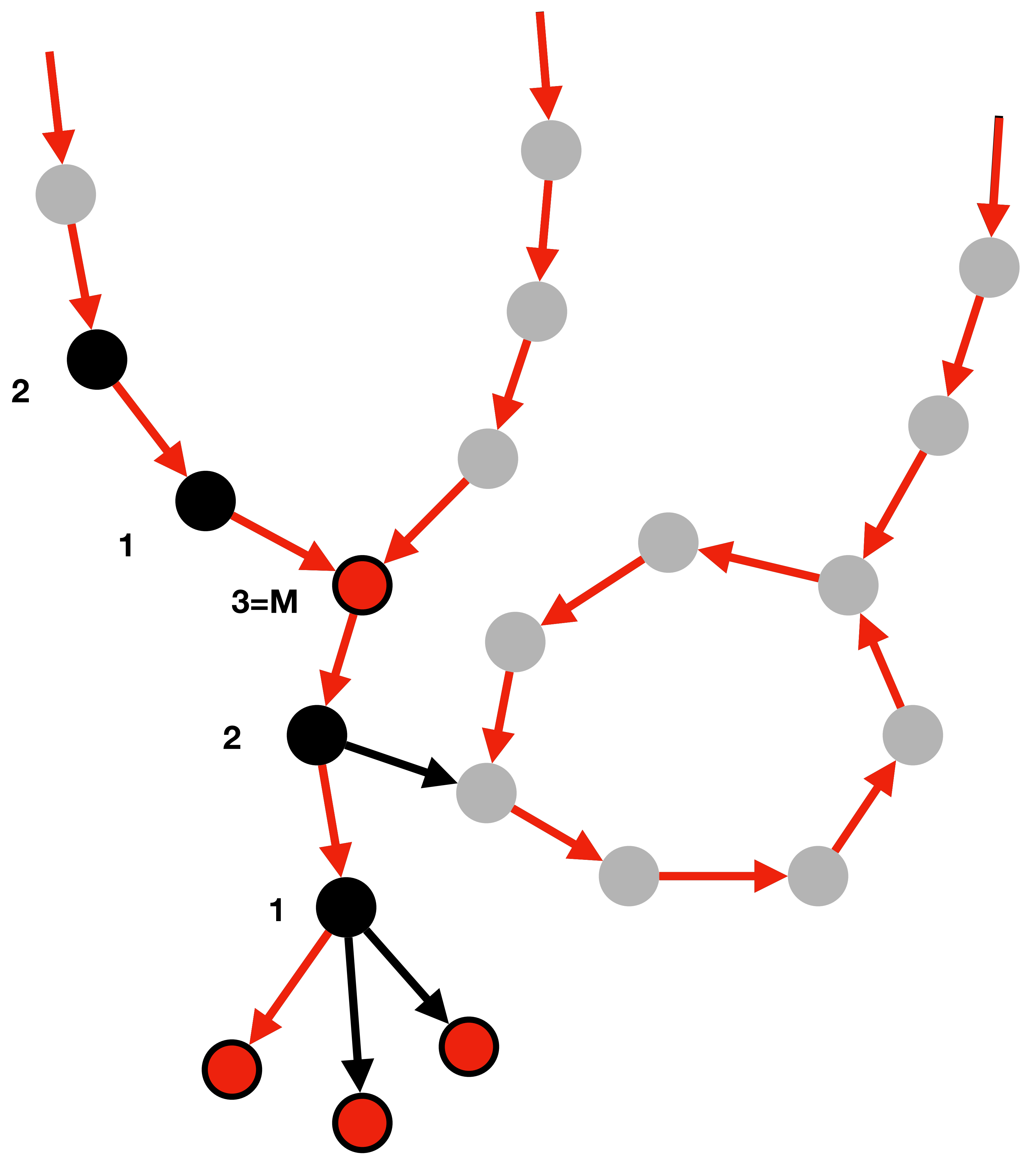
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



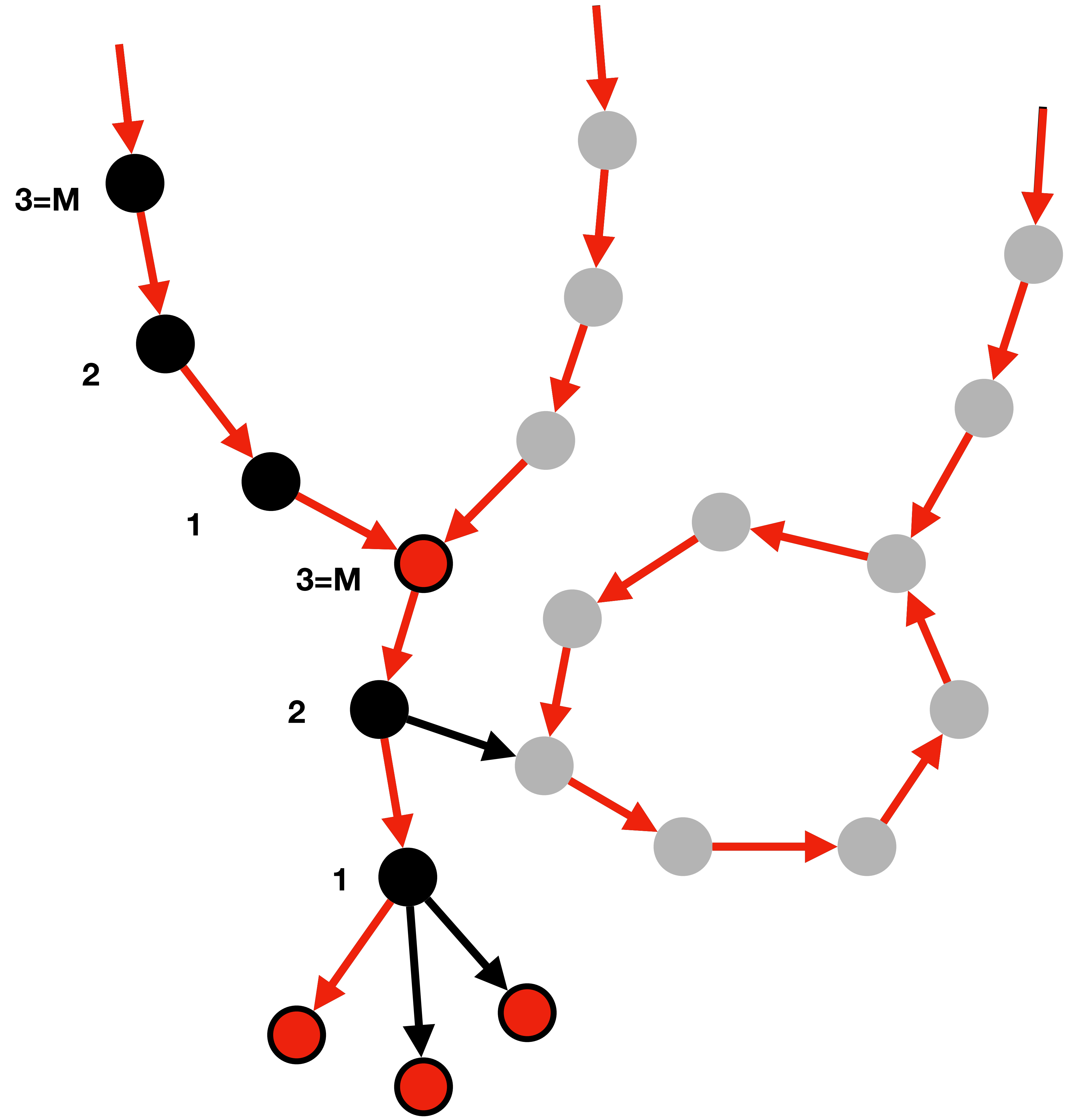
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



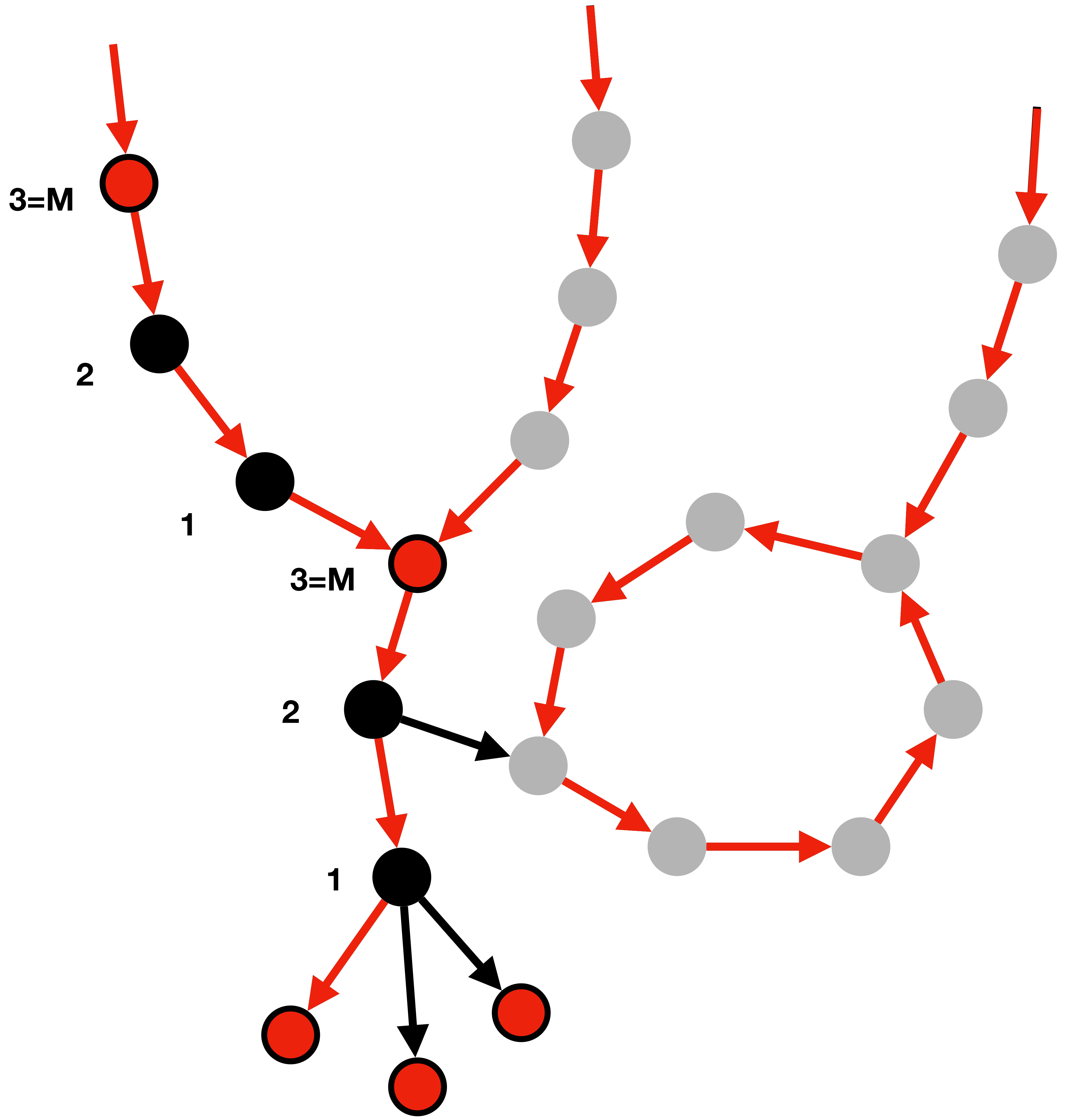
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

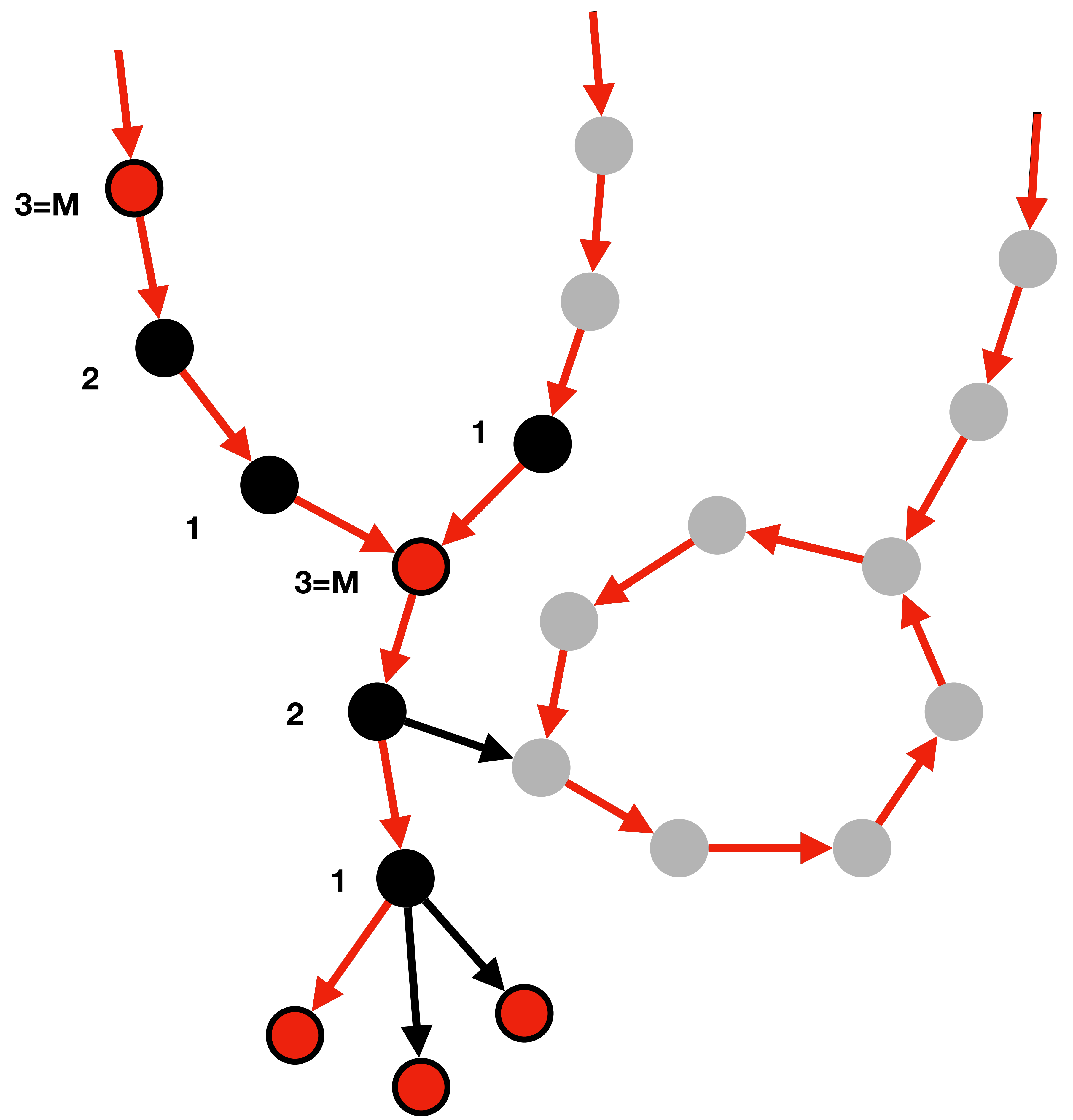
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

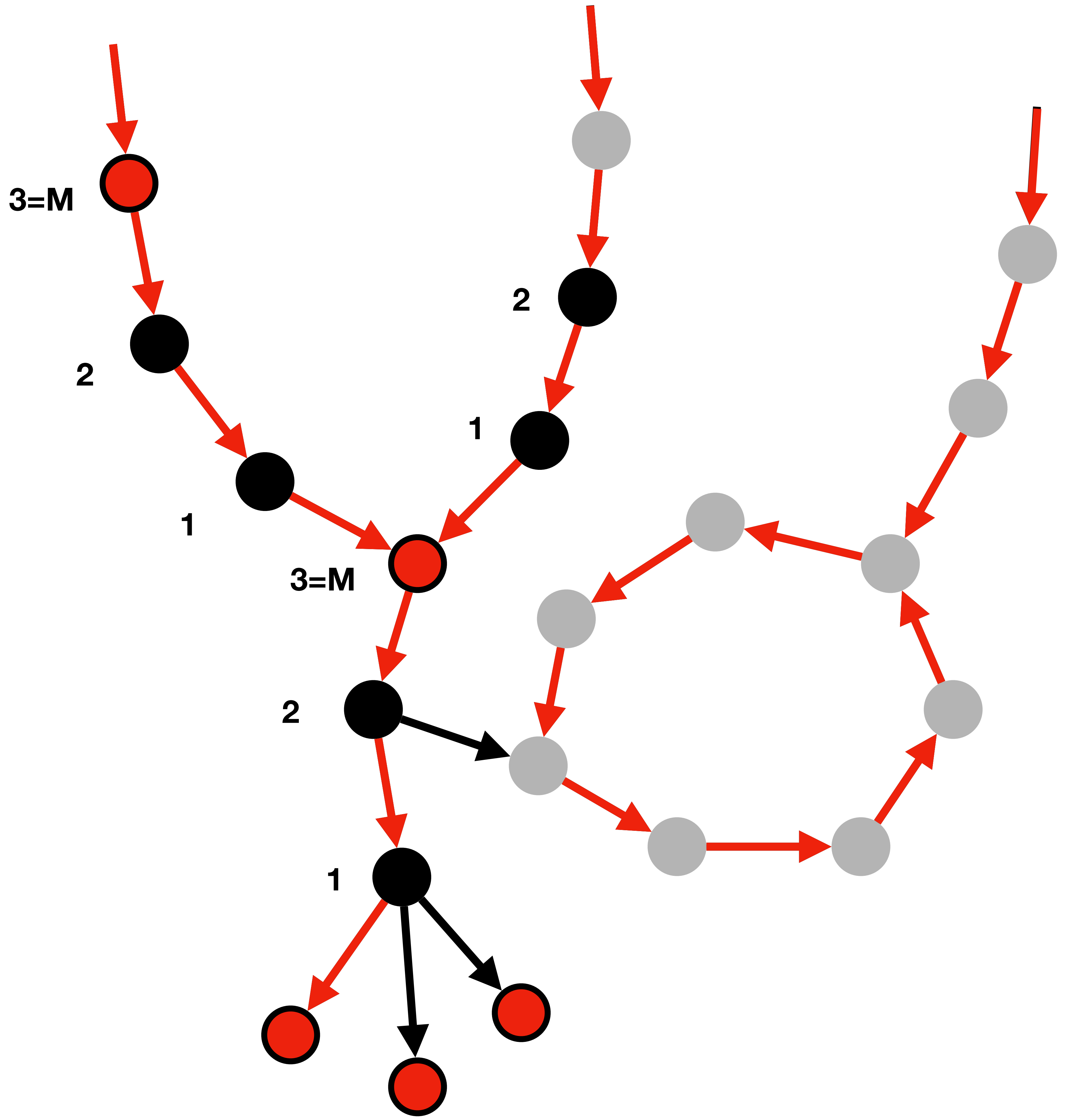
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

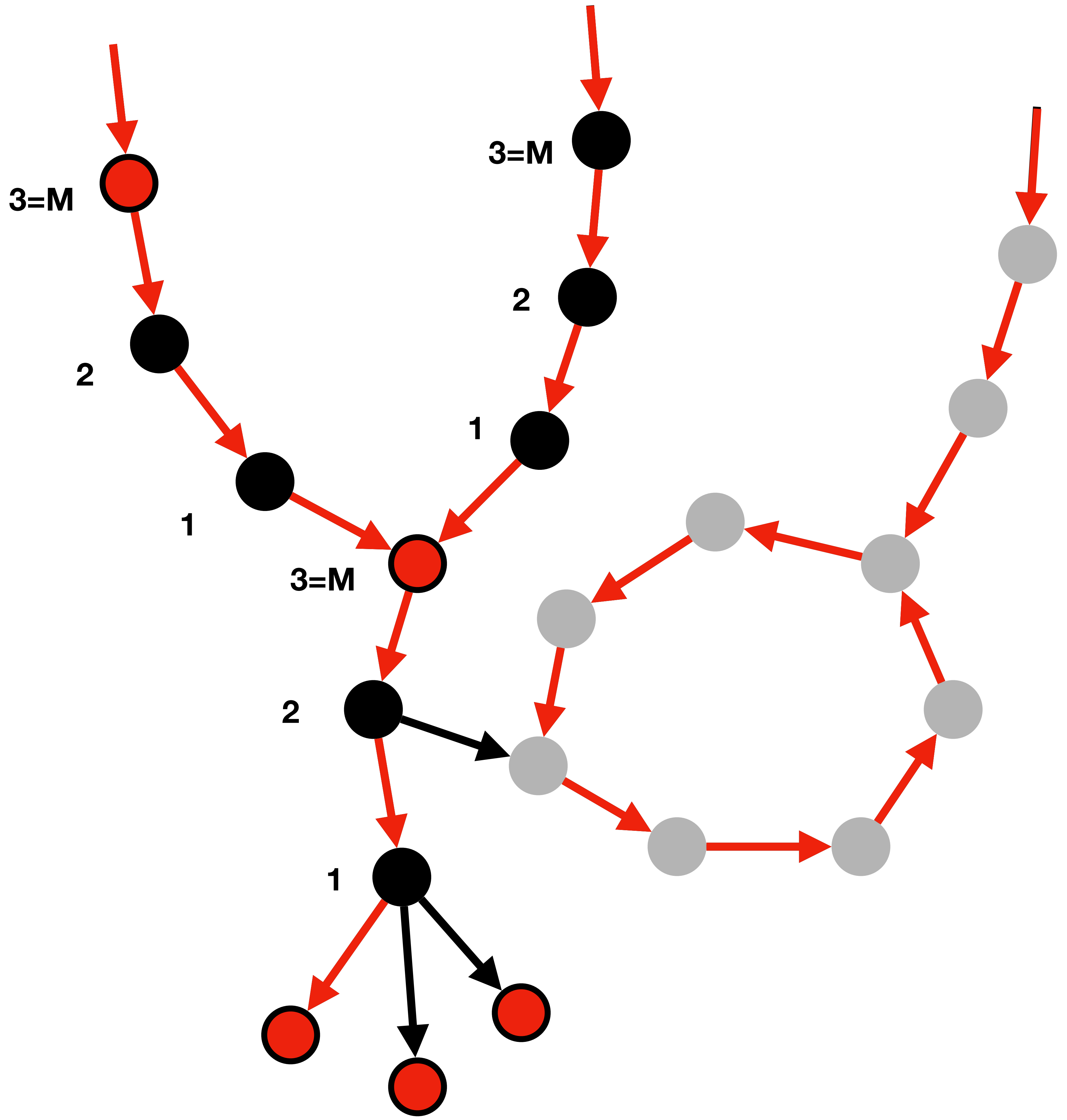
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

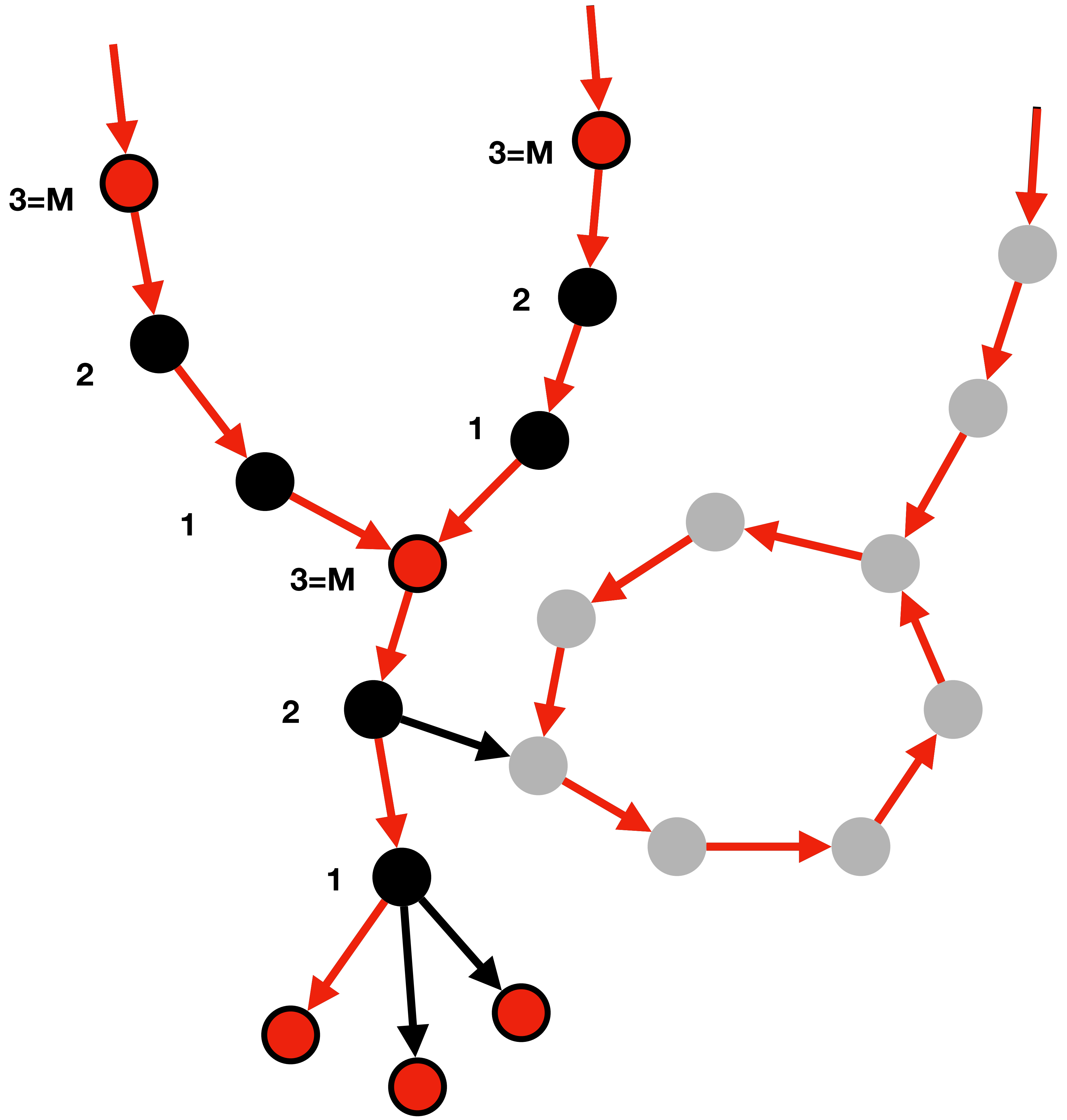
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

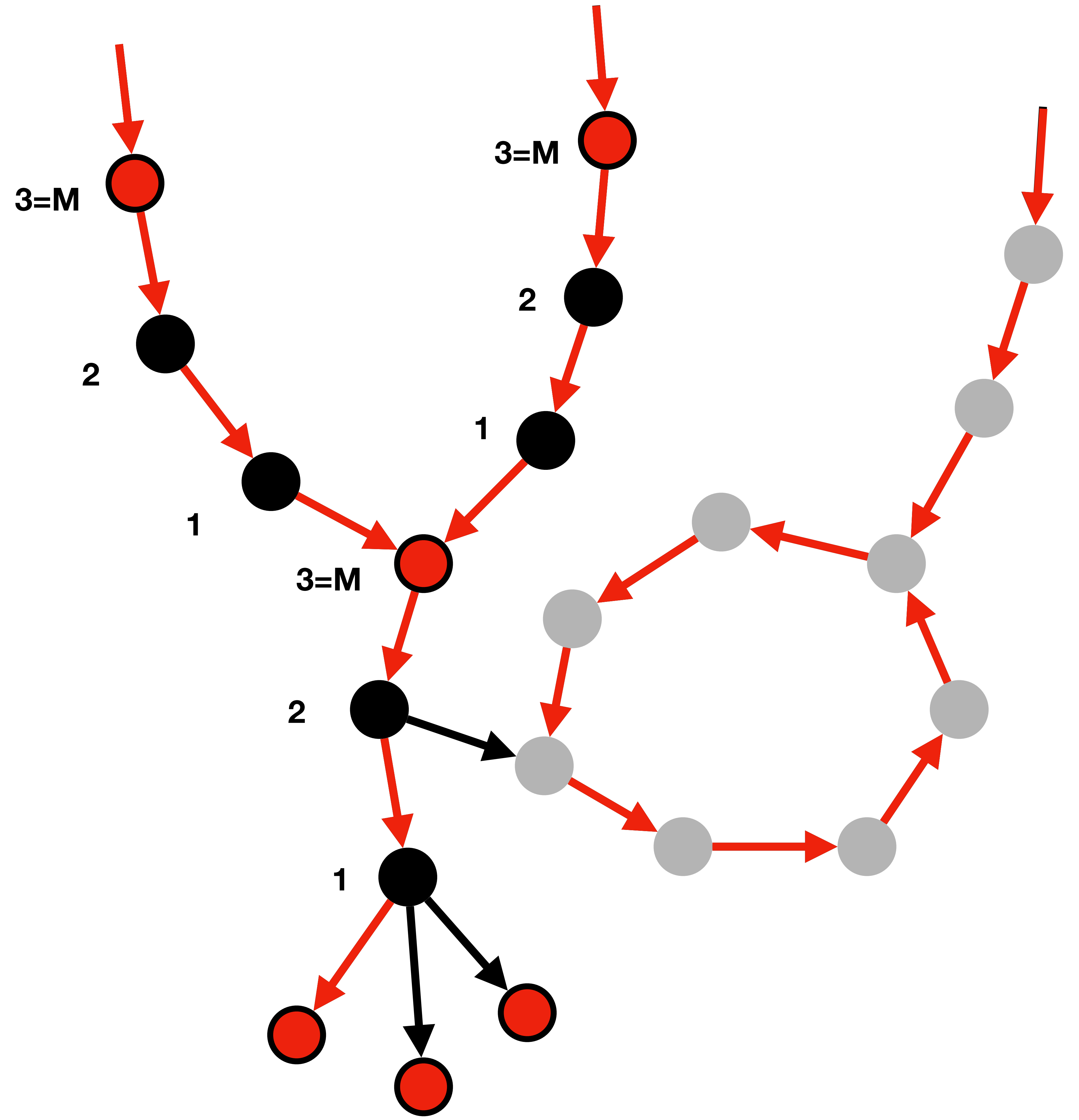
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)



Method

RowDiff: Anchor Assignment

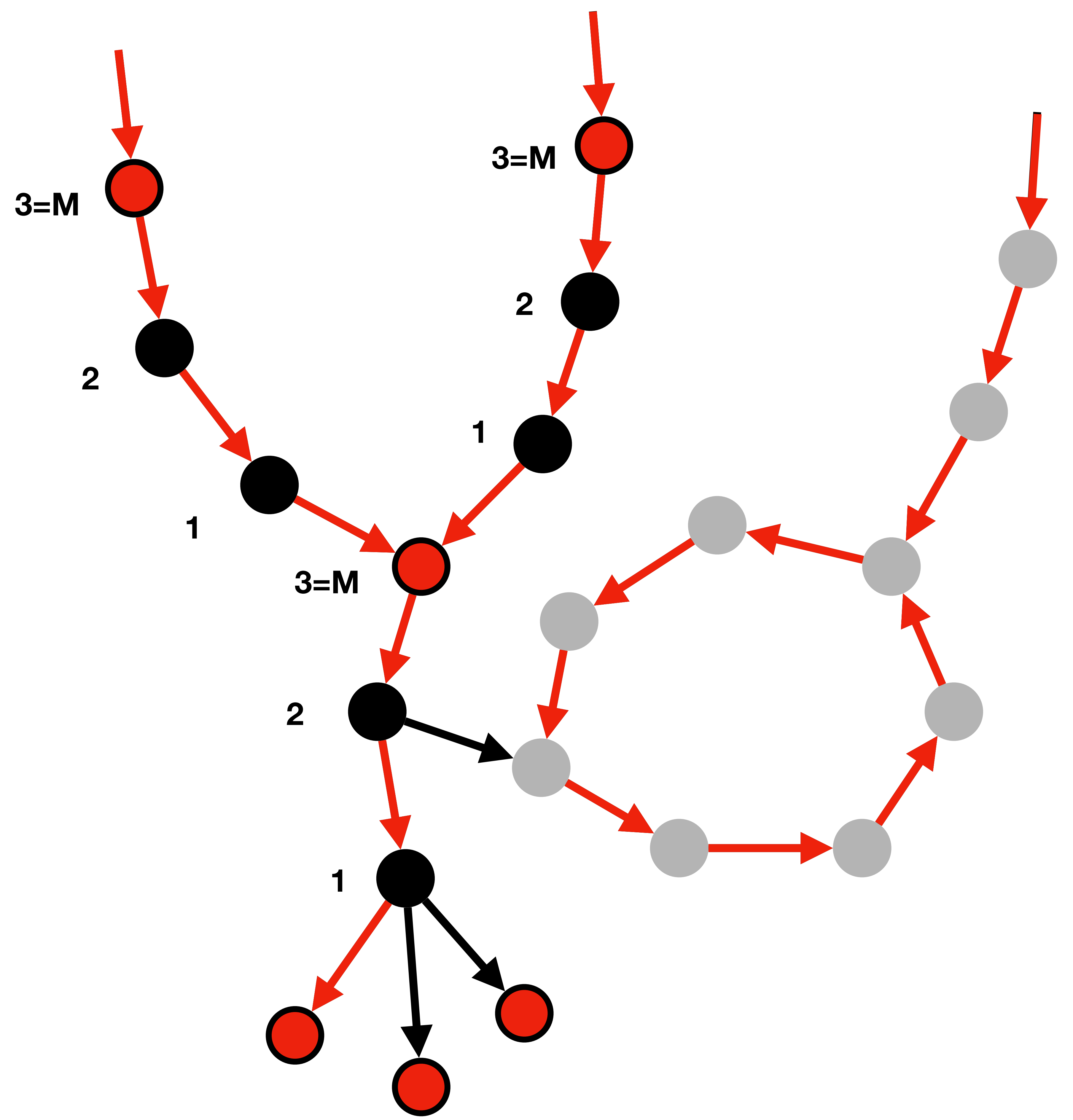
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)
   - Up to this point, there are no row-diff paths longer than M



Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)
    - Up to this point, there are no row-diff paths longer than M
    - In practice, this covers 98% of the nodes



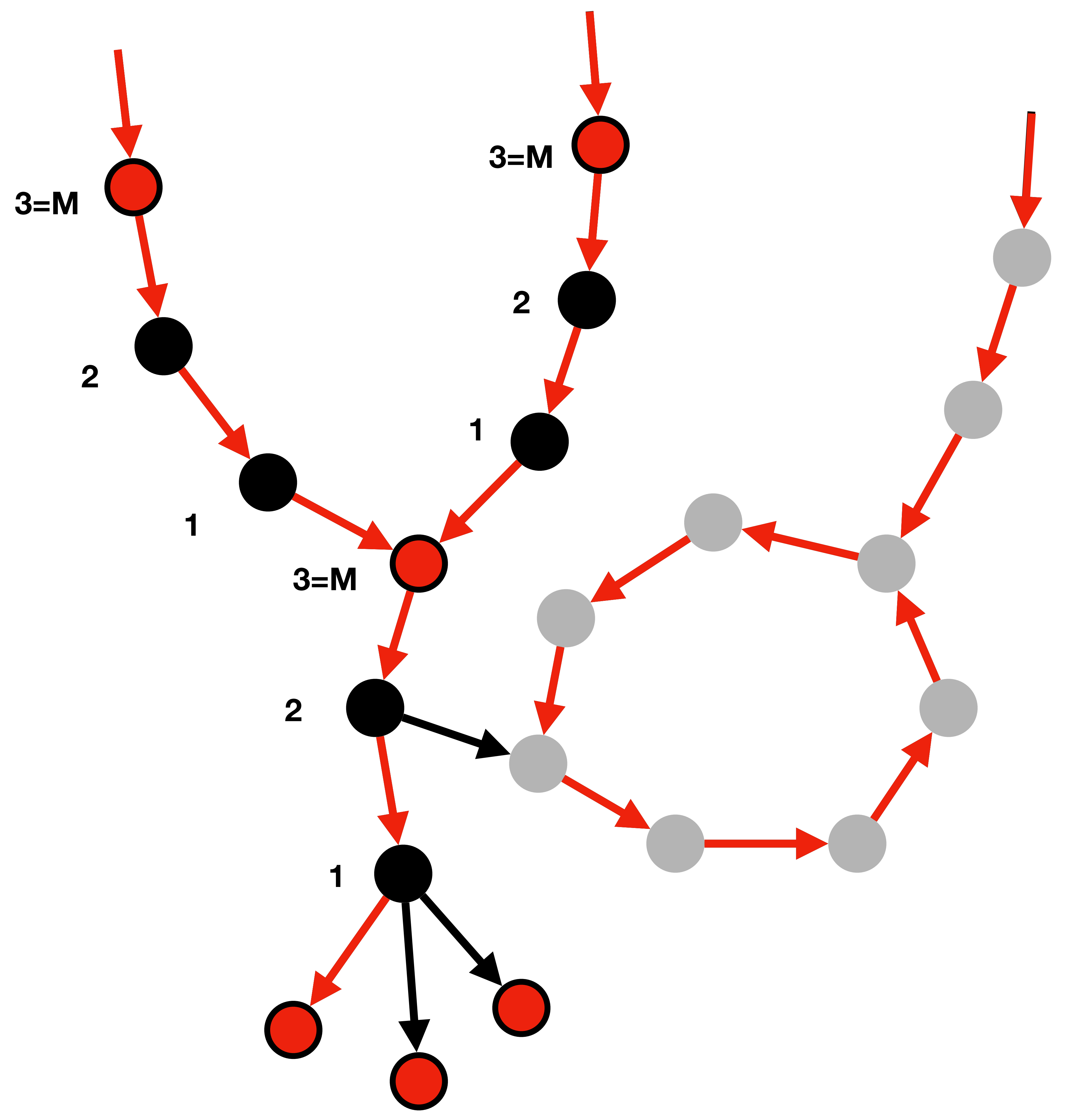
Method

RowDiff: Anchor Assignment

1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)

2. **Anchor** all **sink nodes**

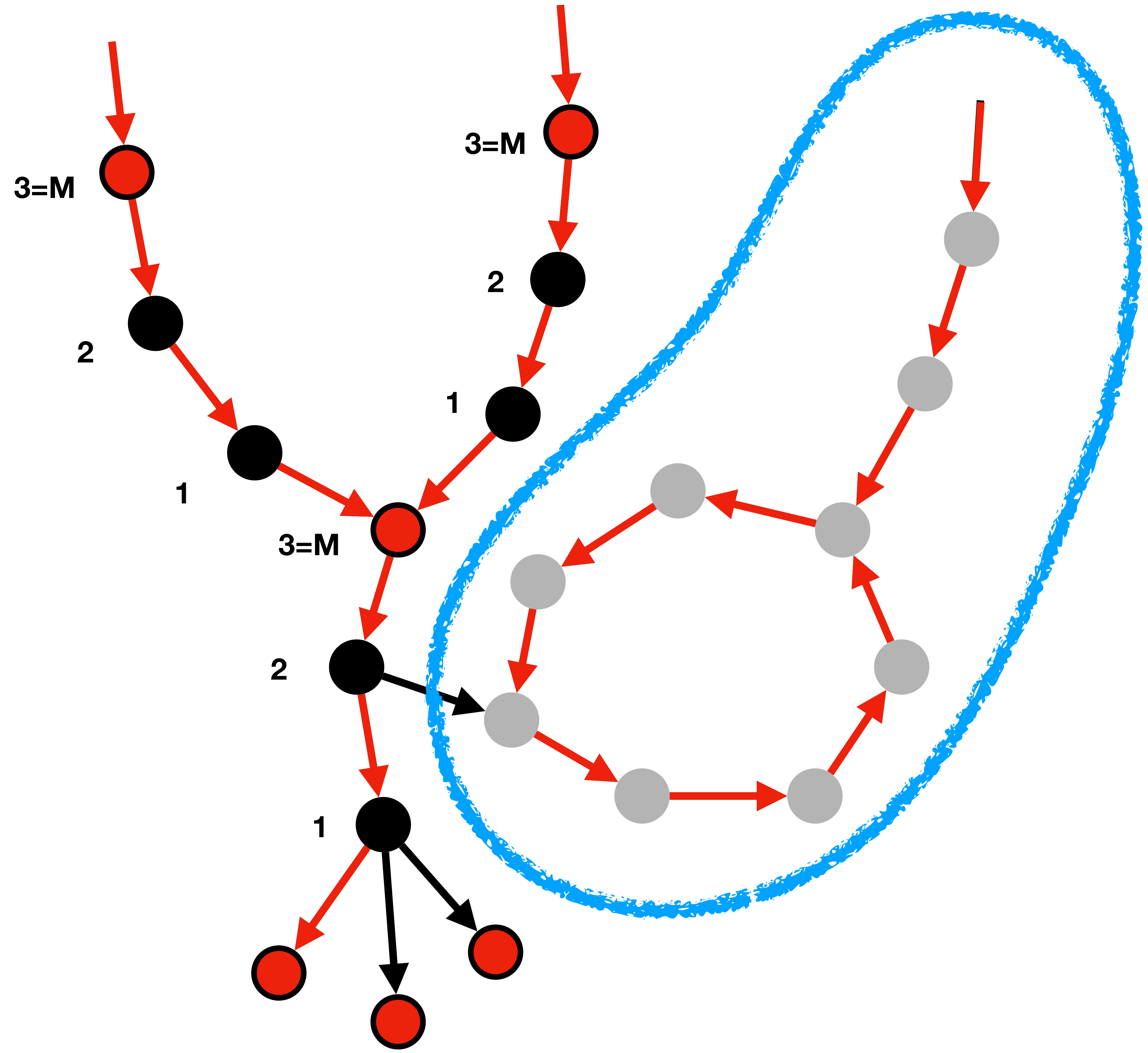
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)
   - Up to this point, there are no row-diff paths longer than M
   - In practice, this covers 98% of the nodes
   - Traverses trees, hence, easy to parallelize



Method

RowDiff: Anchor Assignment

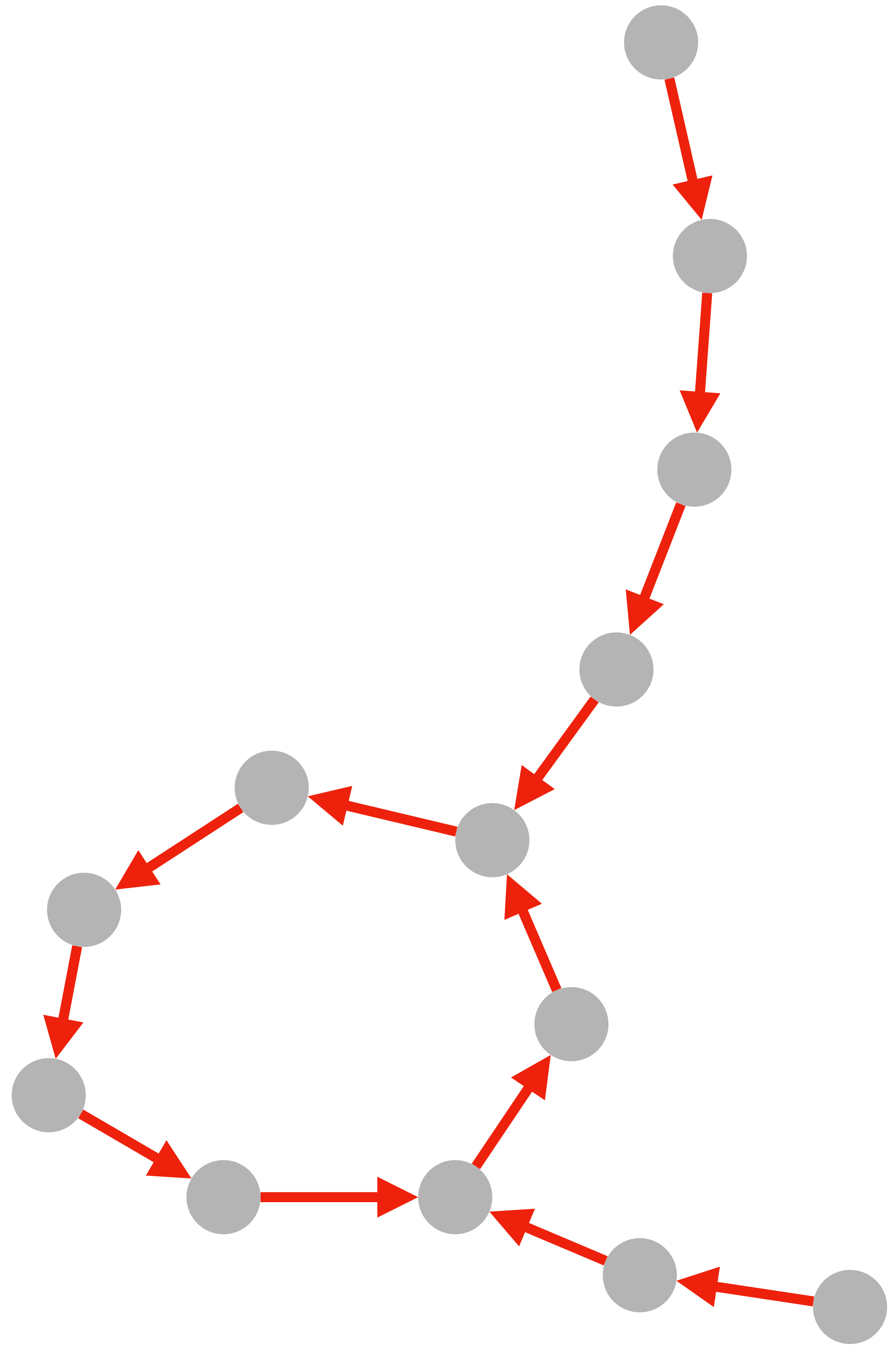
1. For each fork, **pick a row-diff successor** (e.g., lexicographically smallest)
2. **Anchor** all **sink nodes**
3. Start at sink nodes and **traverse** along row-diff paths **backwards** (anchor every M-th node)
   - Up to this point, there are no row-diff paths longer than M
   - In practice, this covers 98% of the nodes
   - Traverses trees, hence, easy to parallelize
4. Now we need to process the **rest** — row-diff paths **that end with a cycle** (forward traversal algorithm, see next slide…)



Method

RowDiff: Anchor Assignment (part 2)

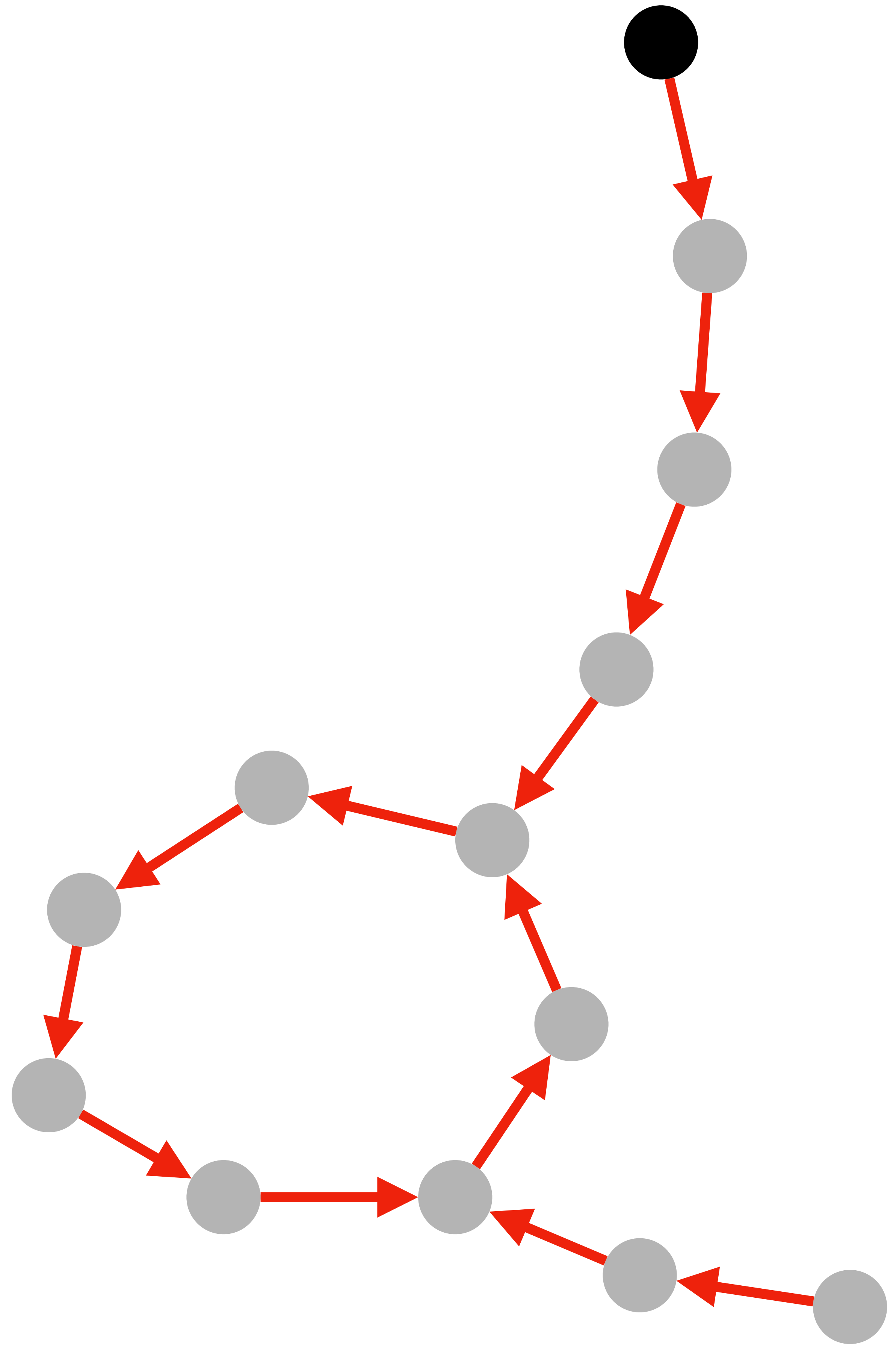
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

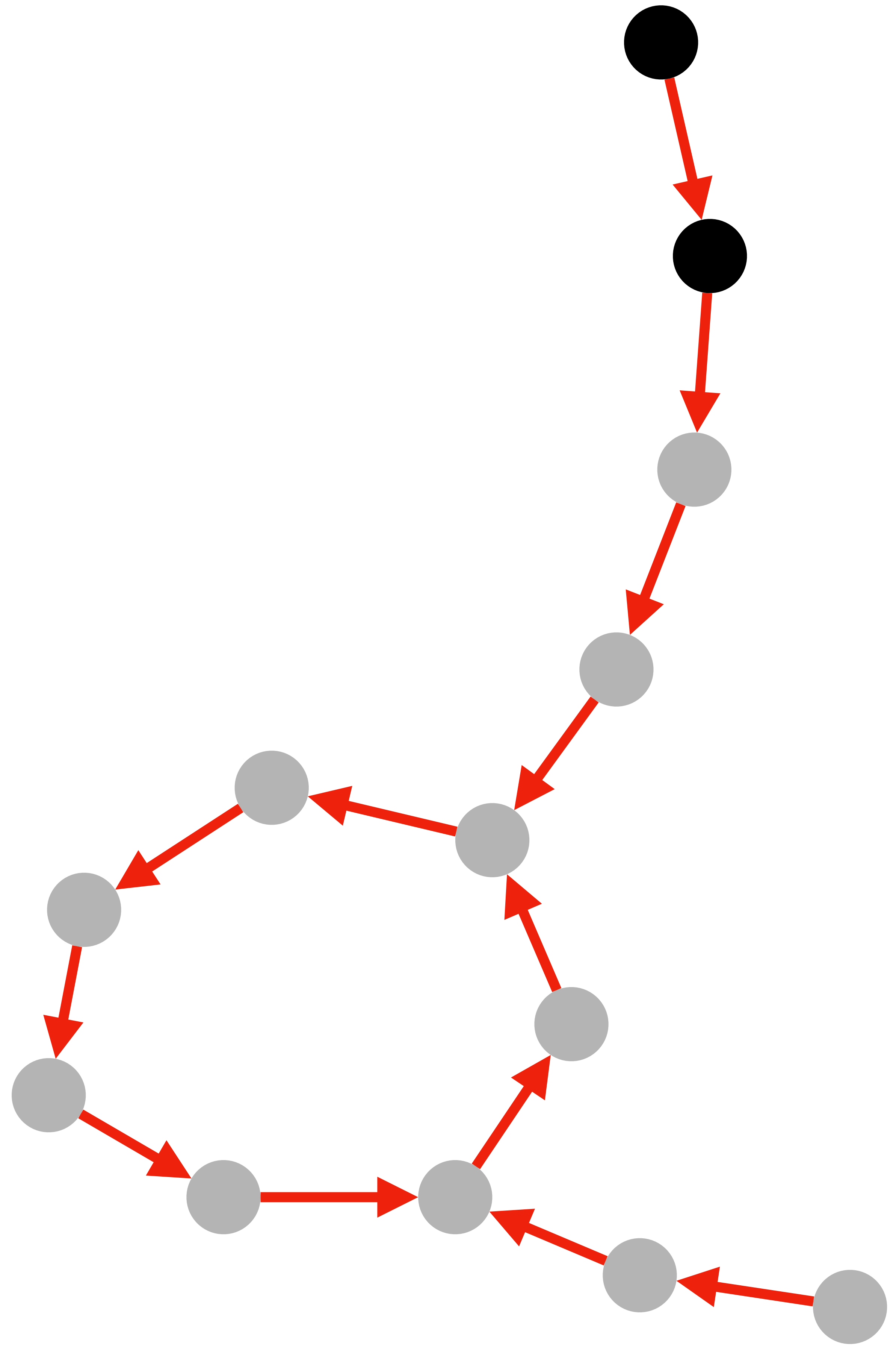
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

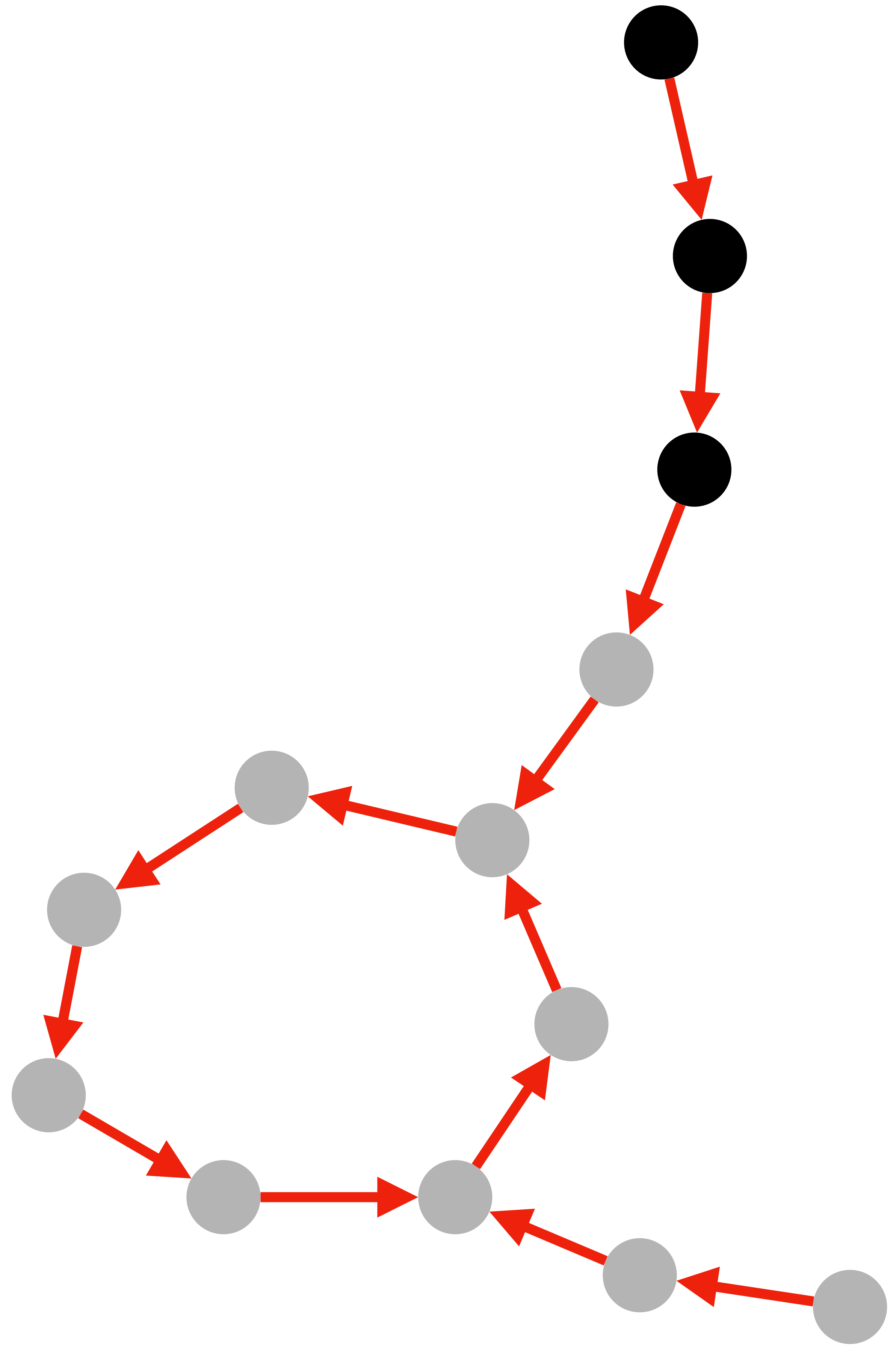
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

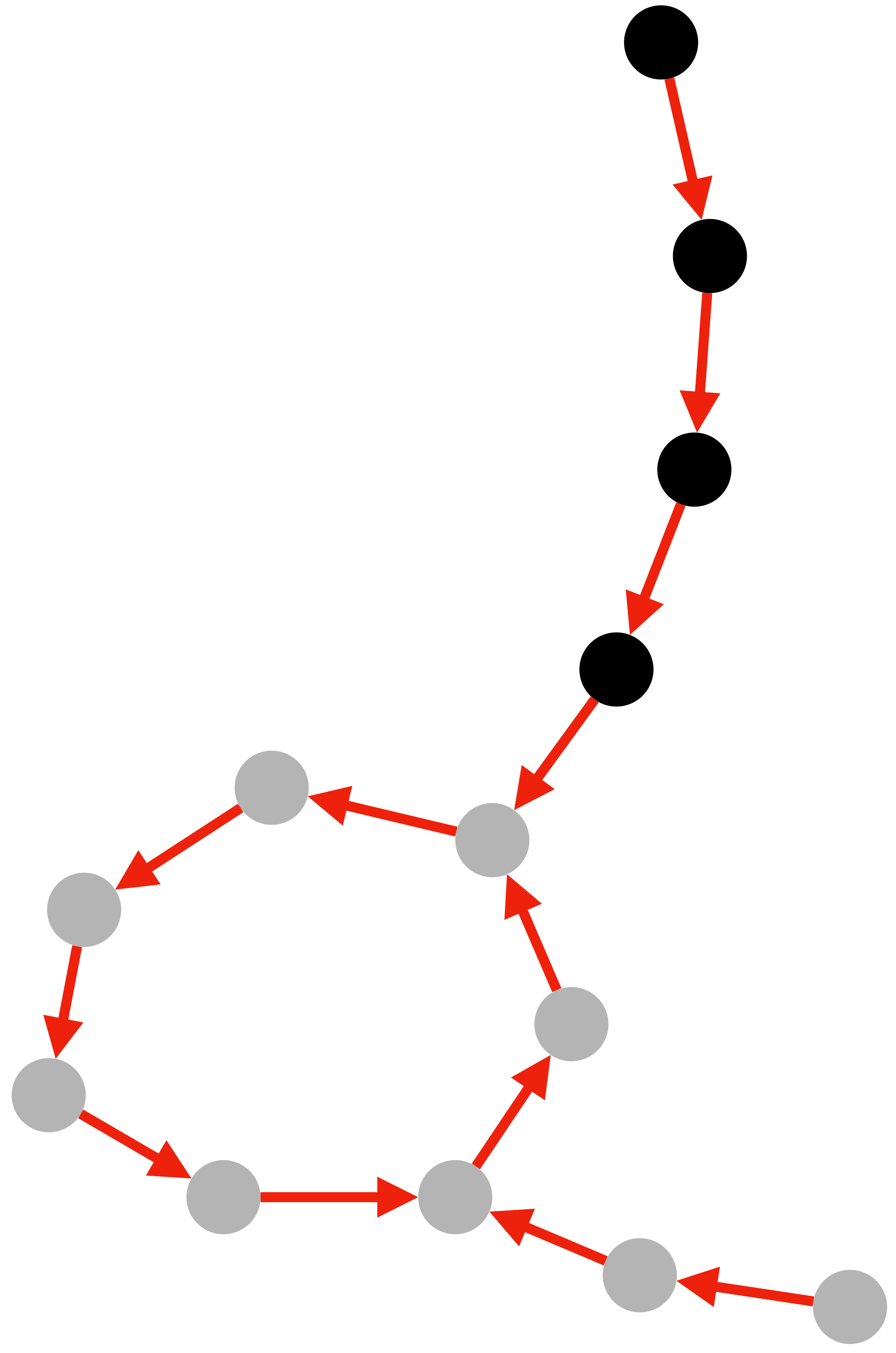
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

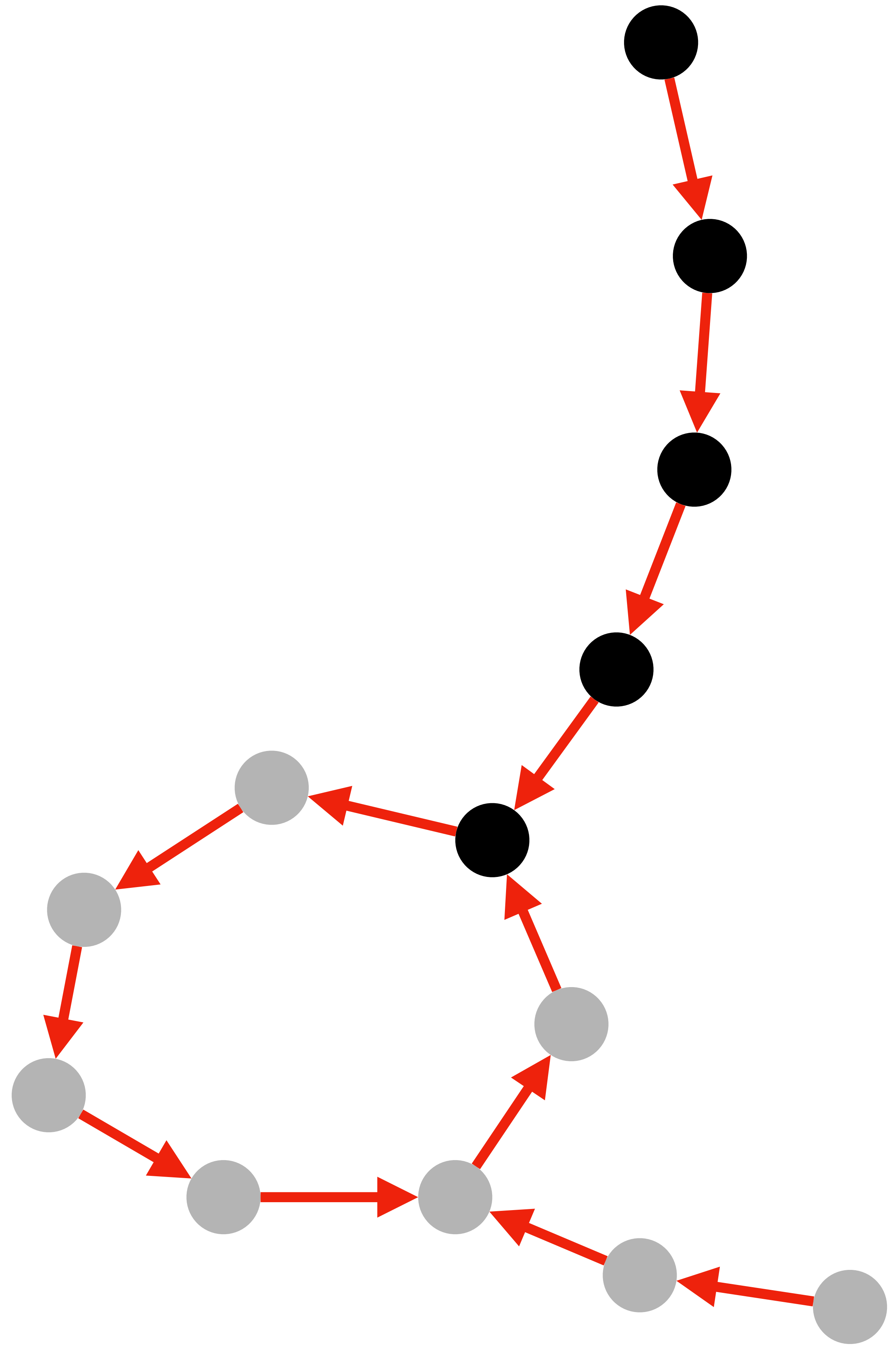
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

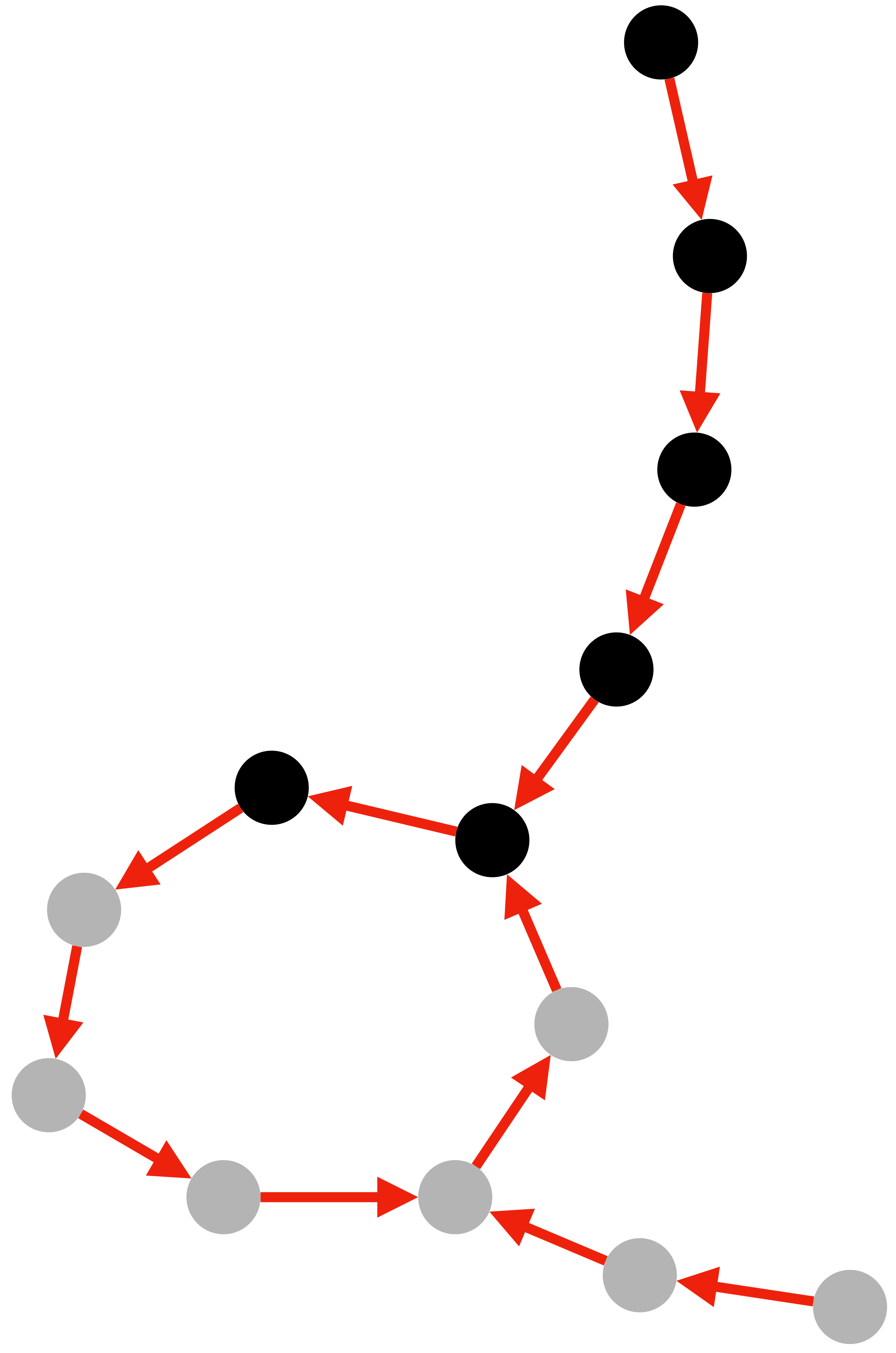
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

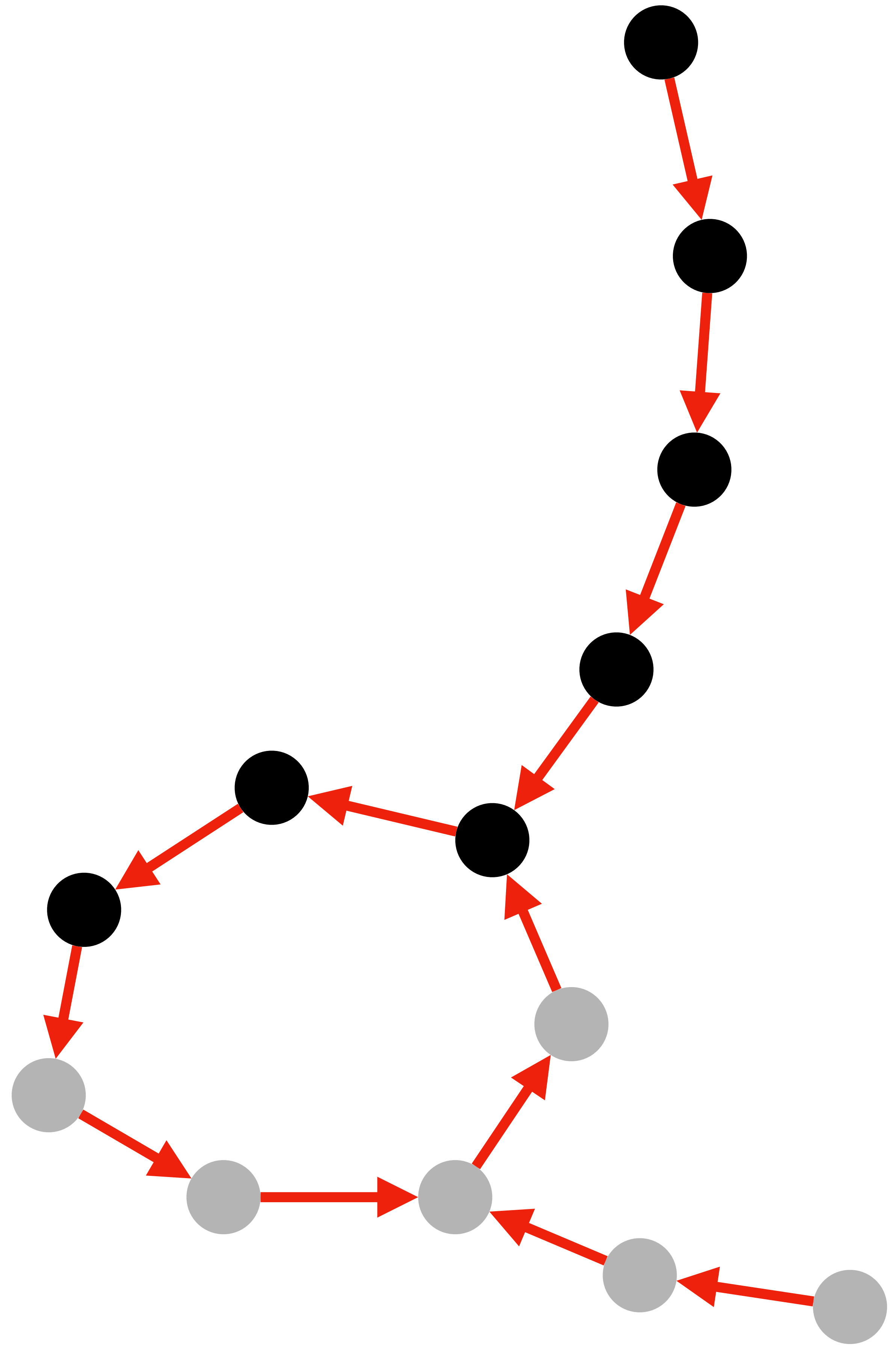
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

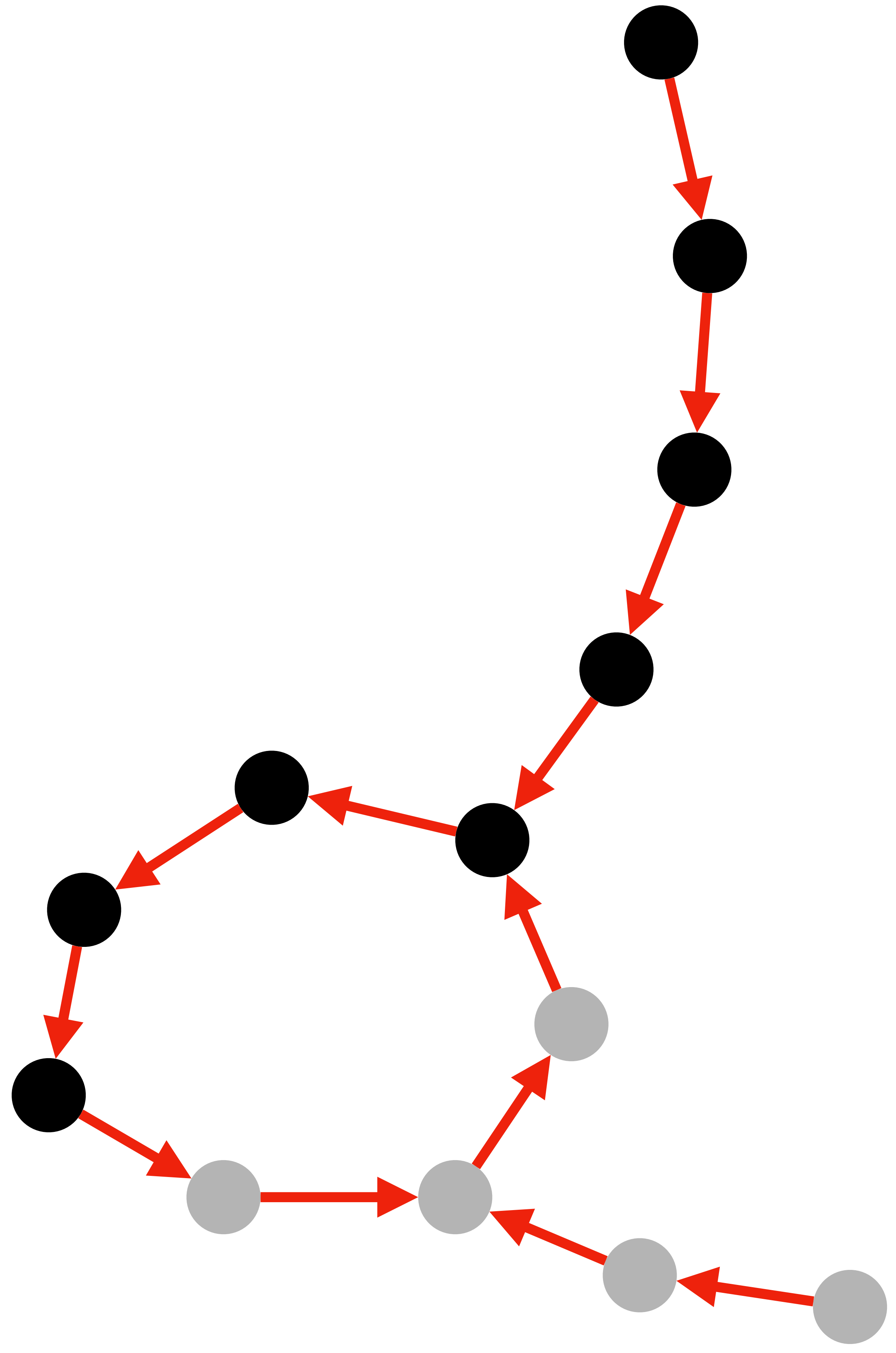
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

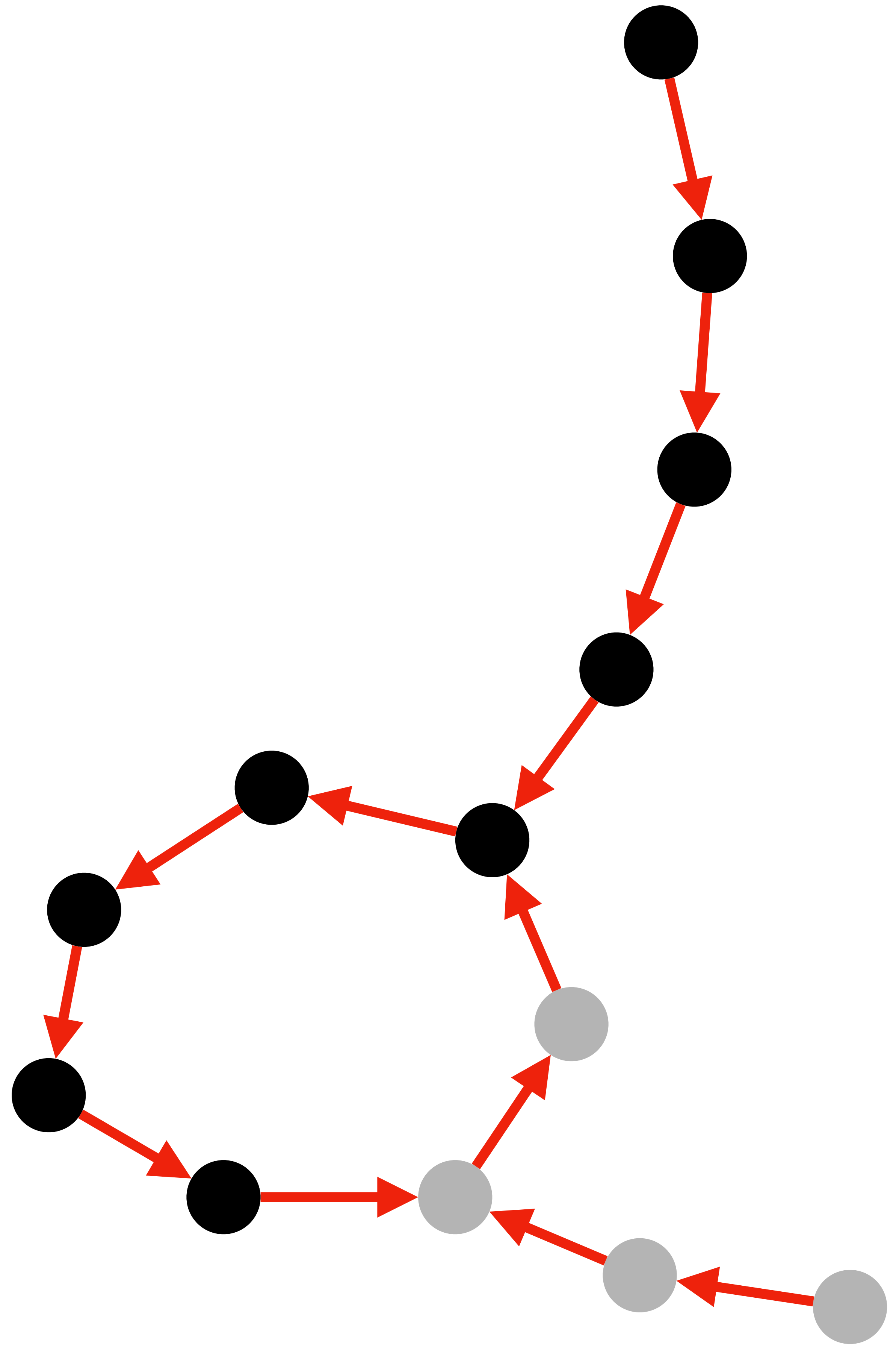
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

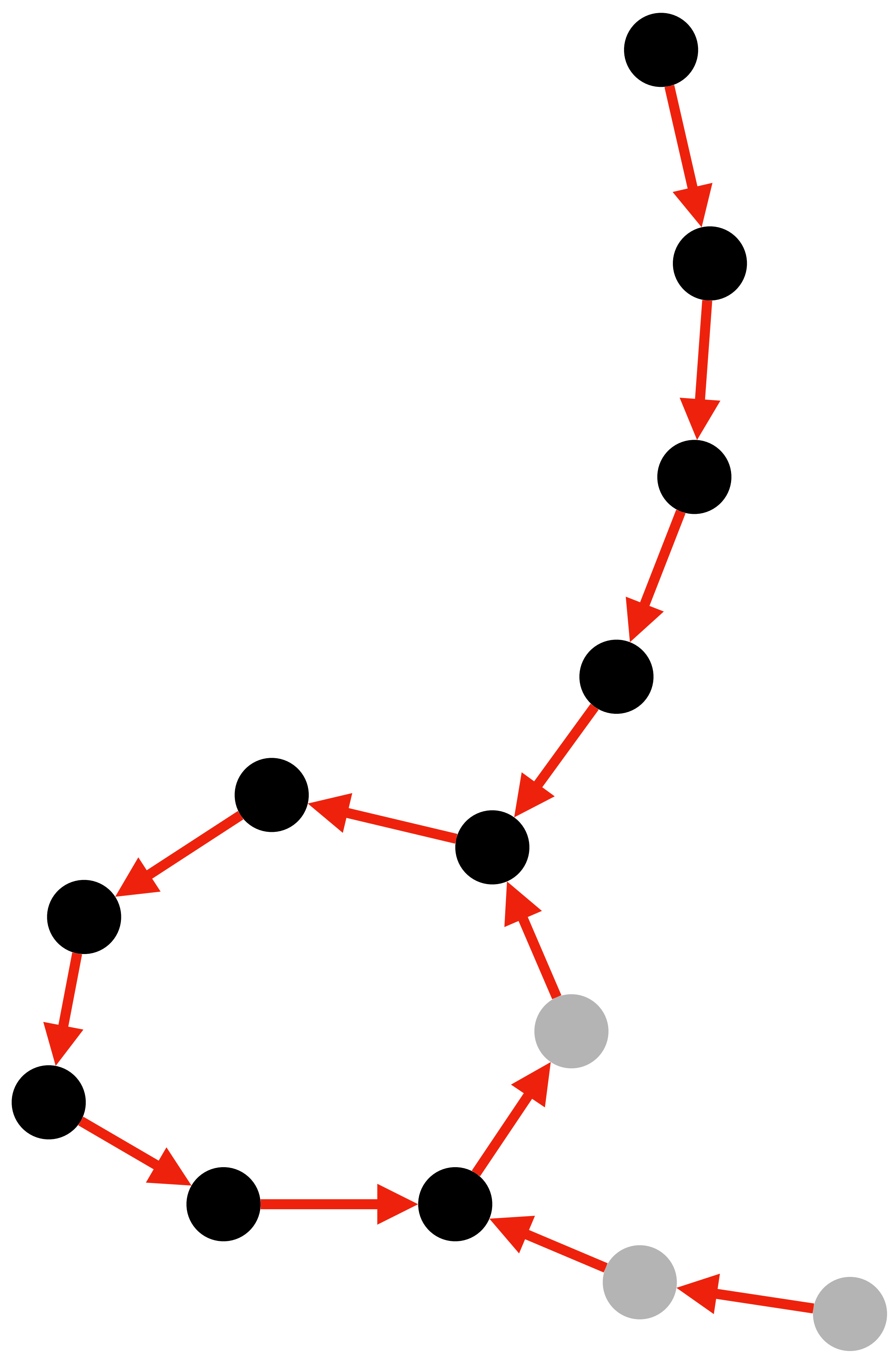
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

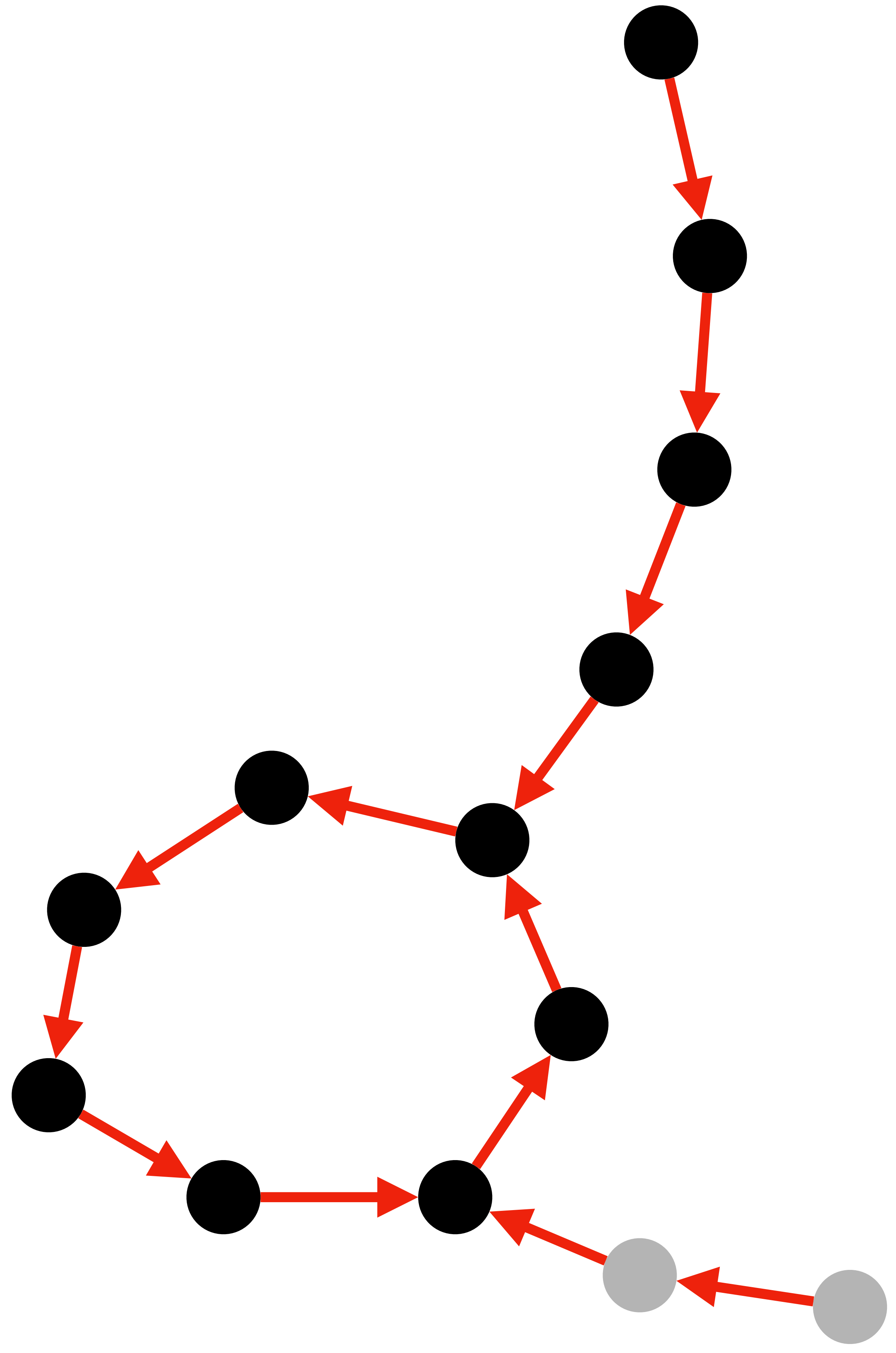
1) Start traversal at unvisited nodes and traverse forward until a visited node is reached



Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

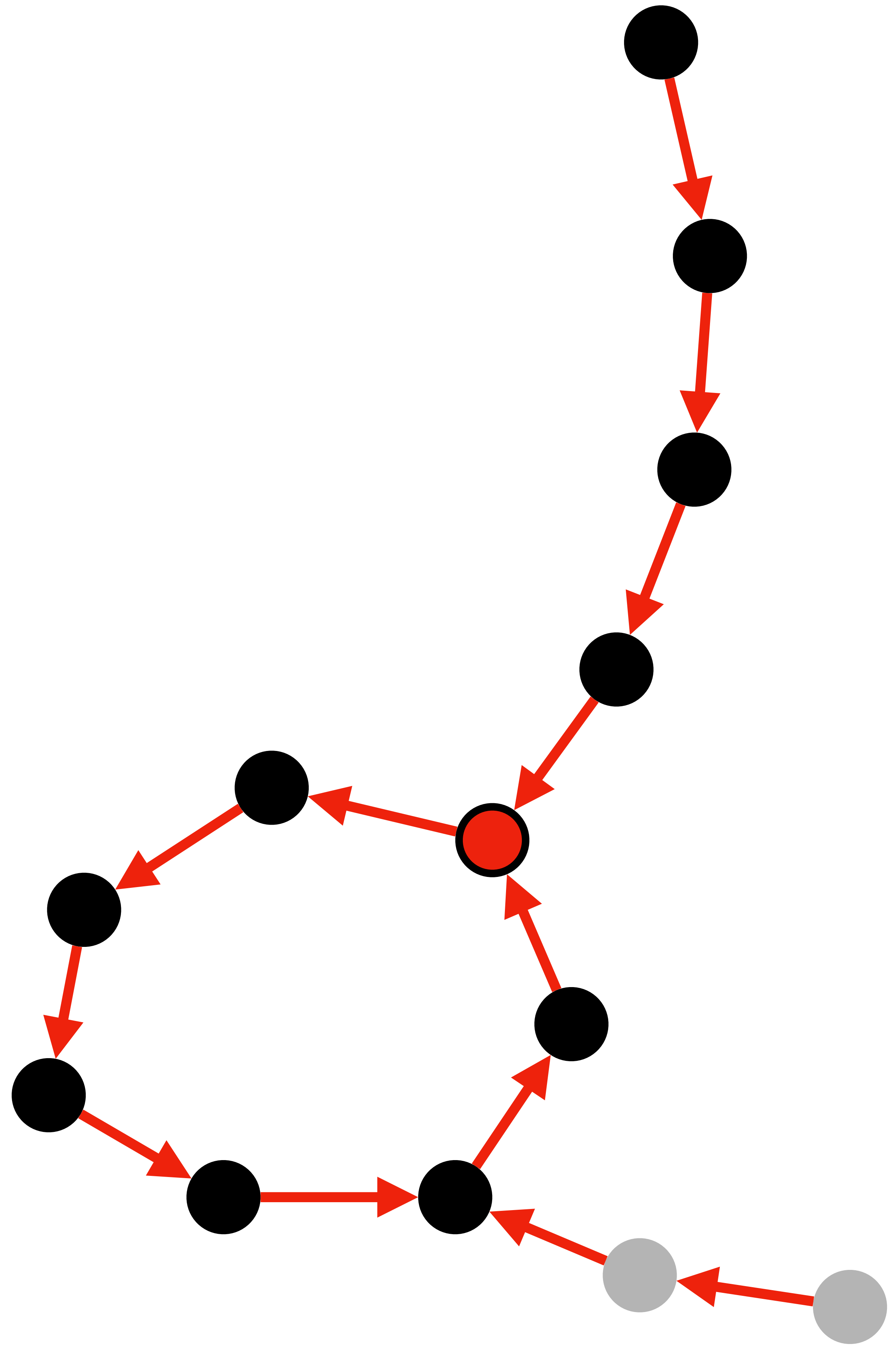


Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

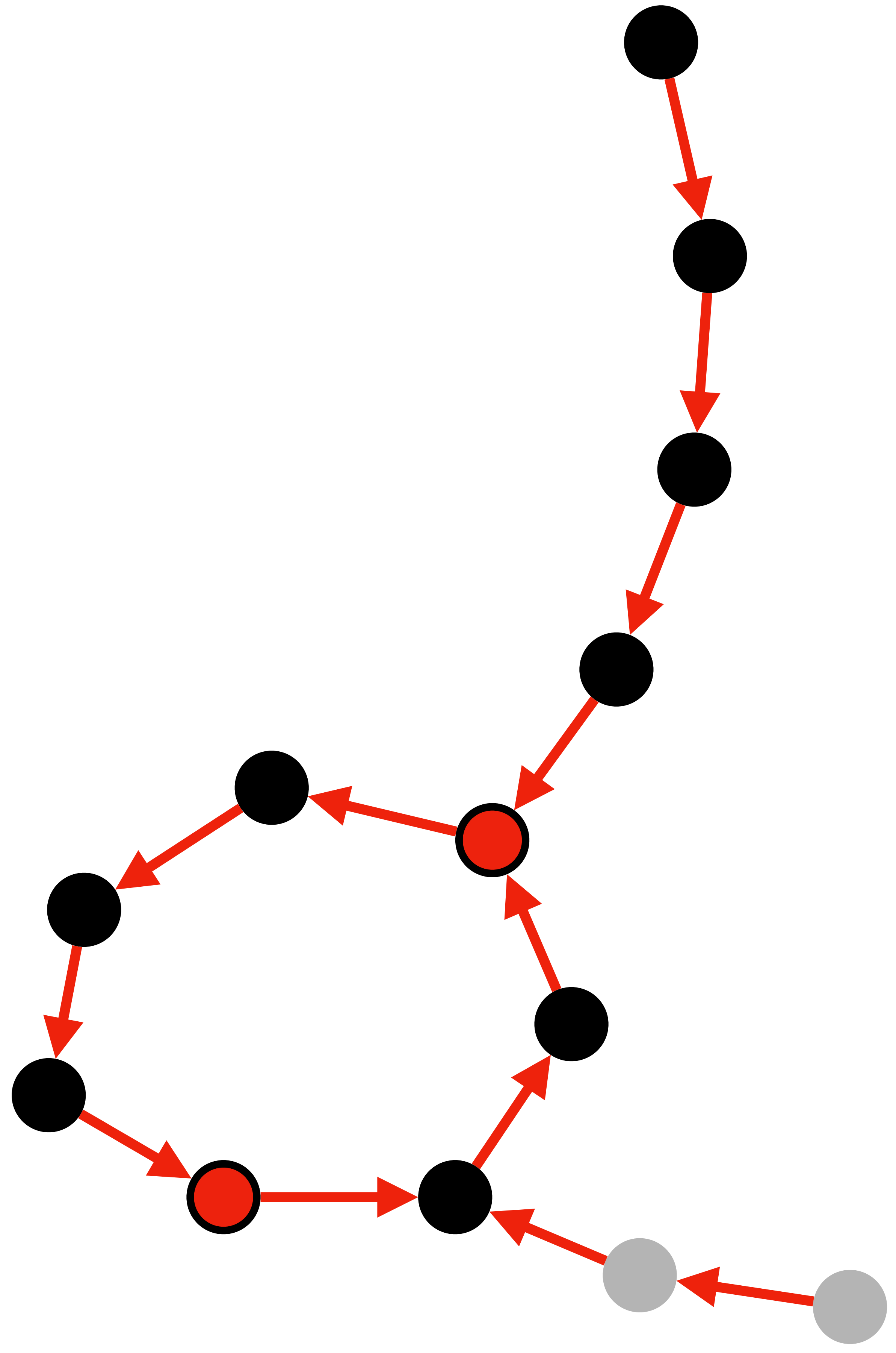


Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

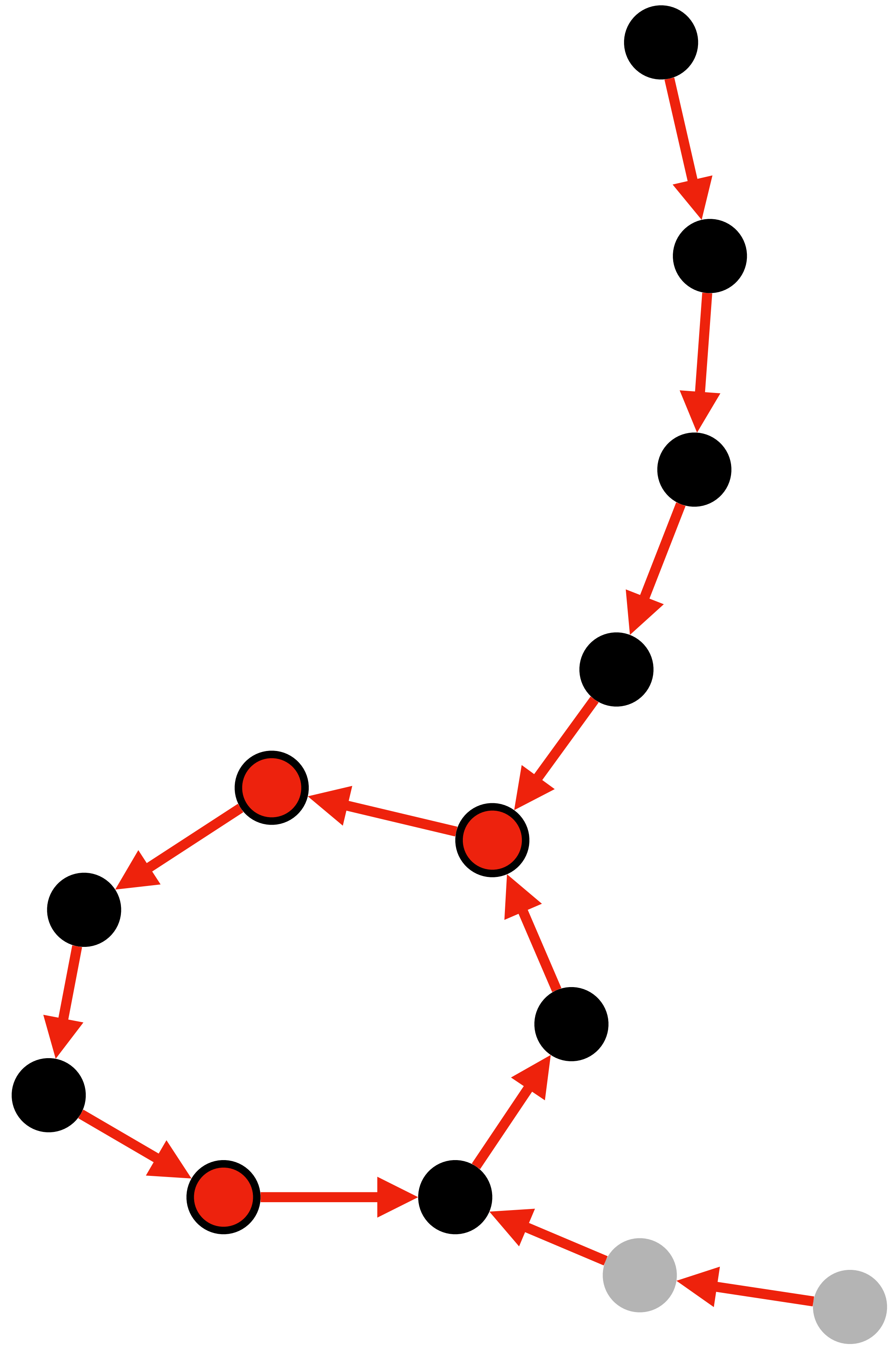


Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

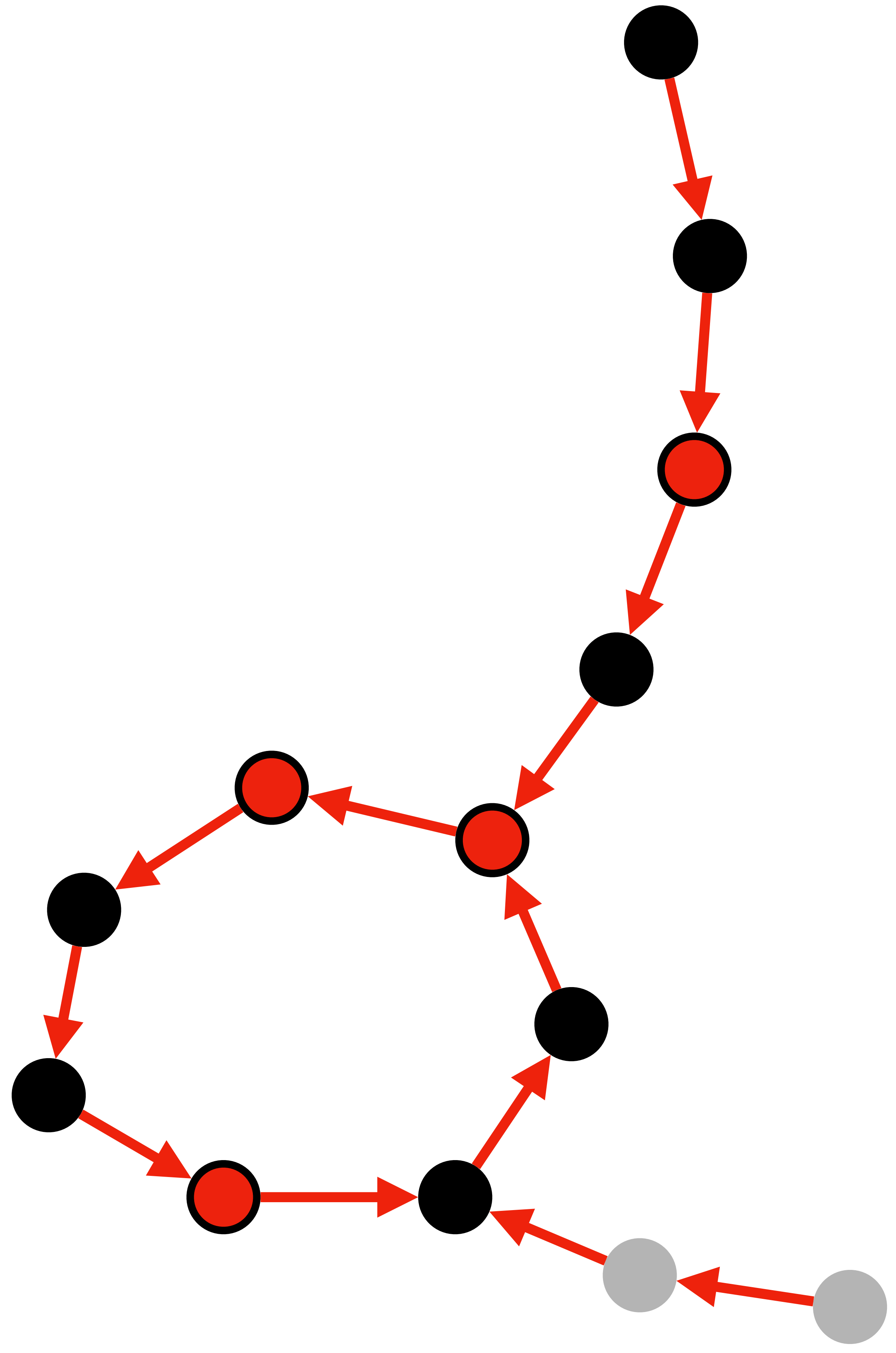


Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)



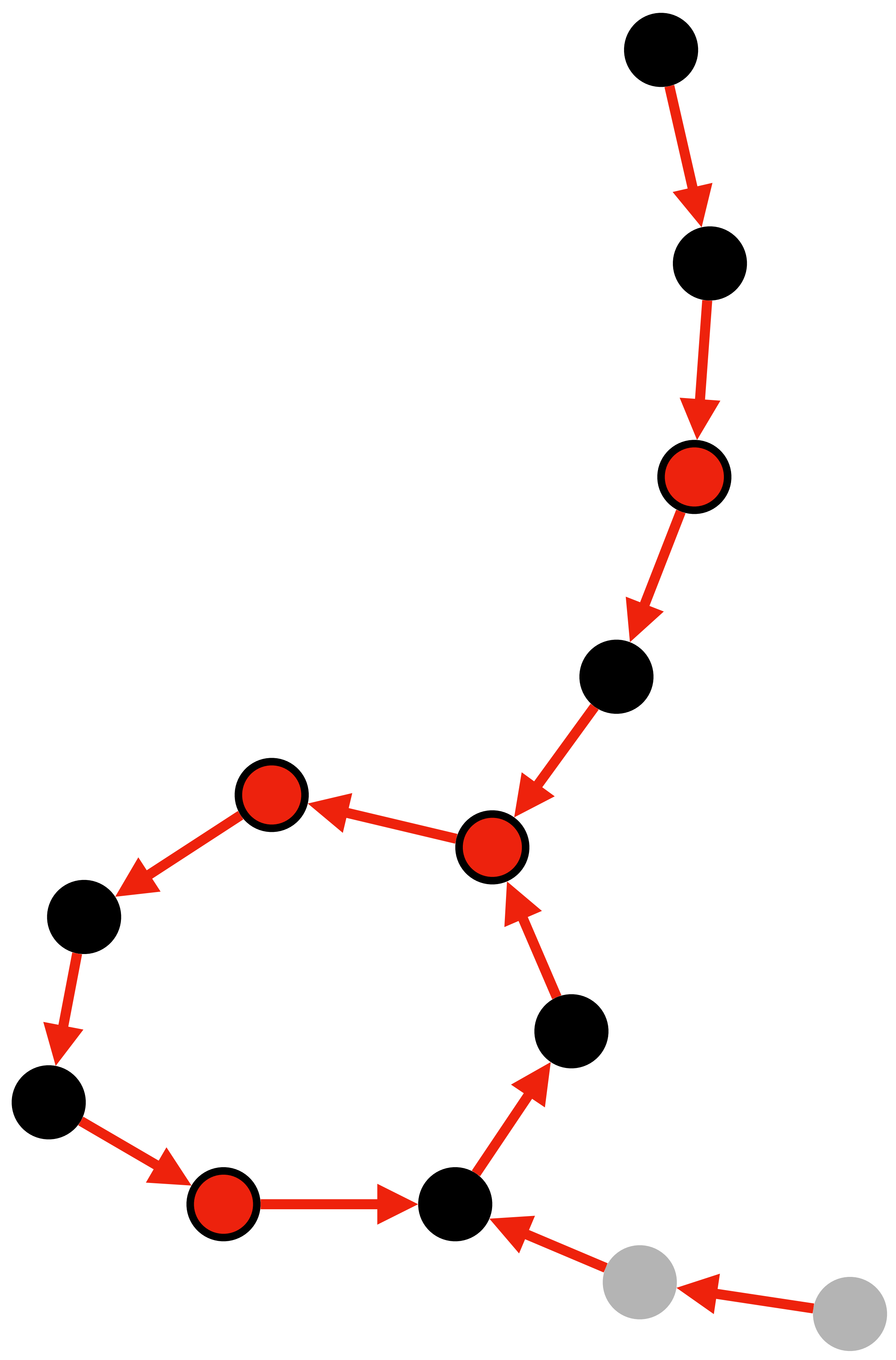
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



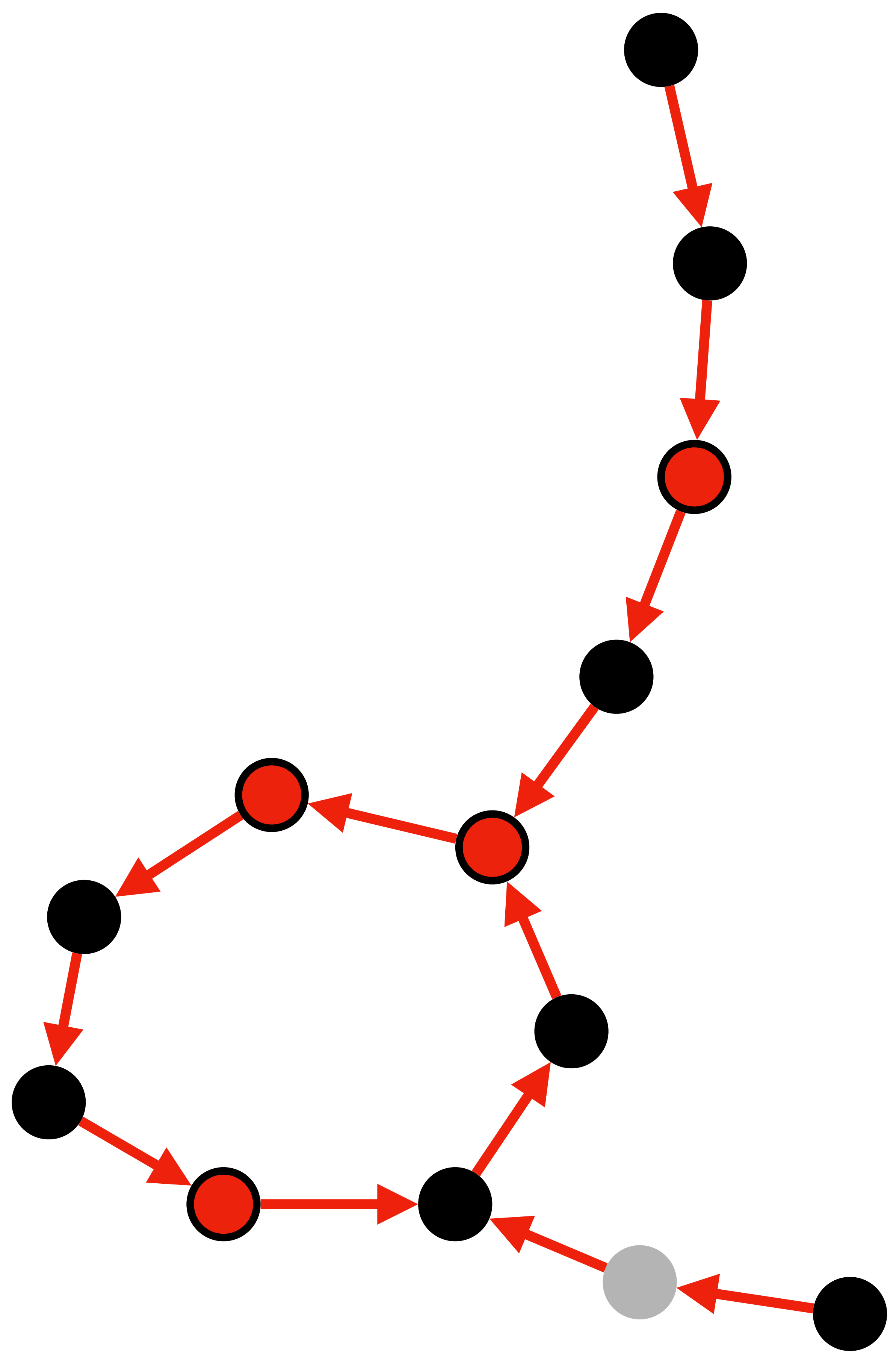
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



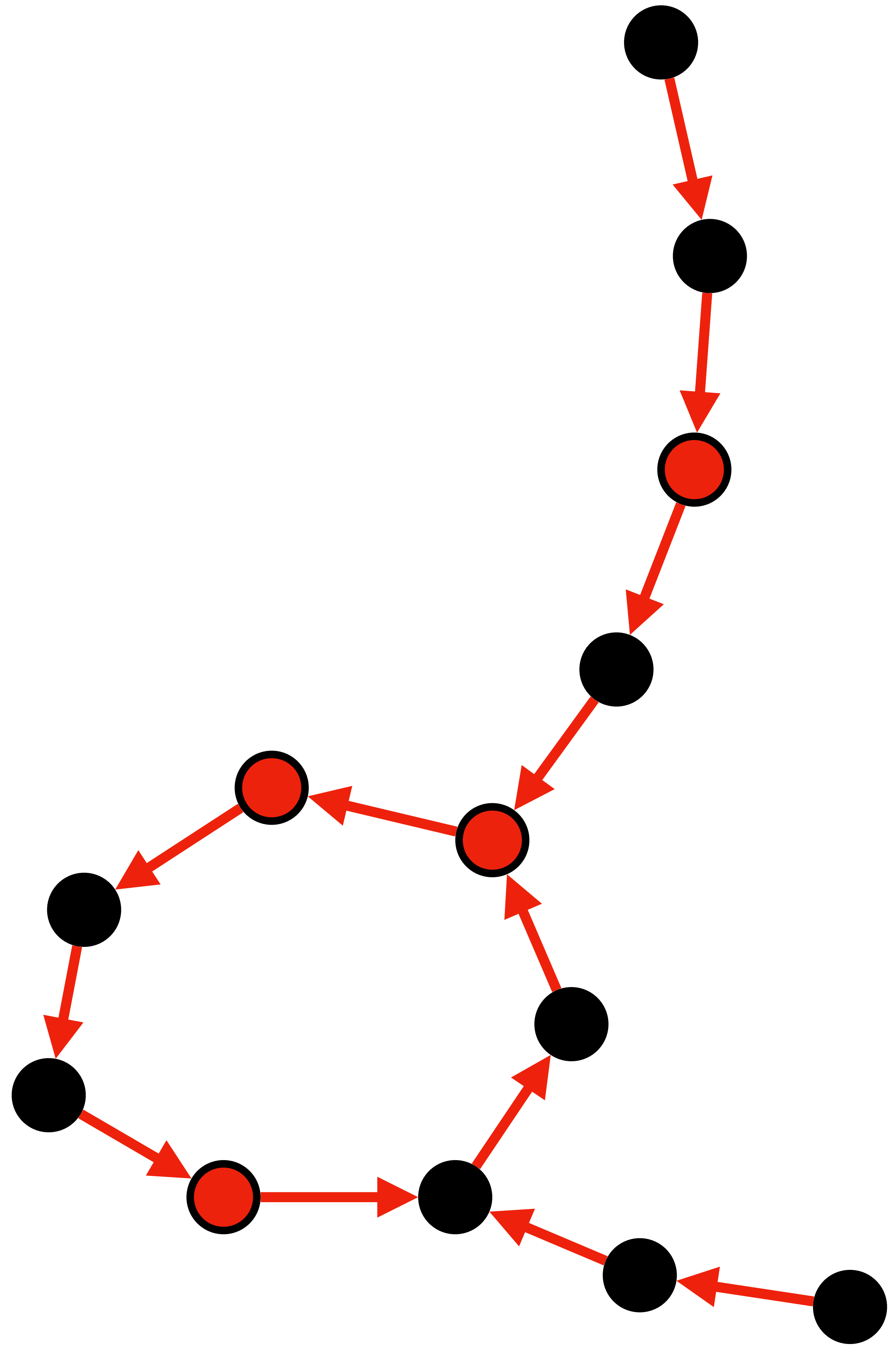
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



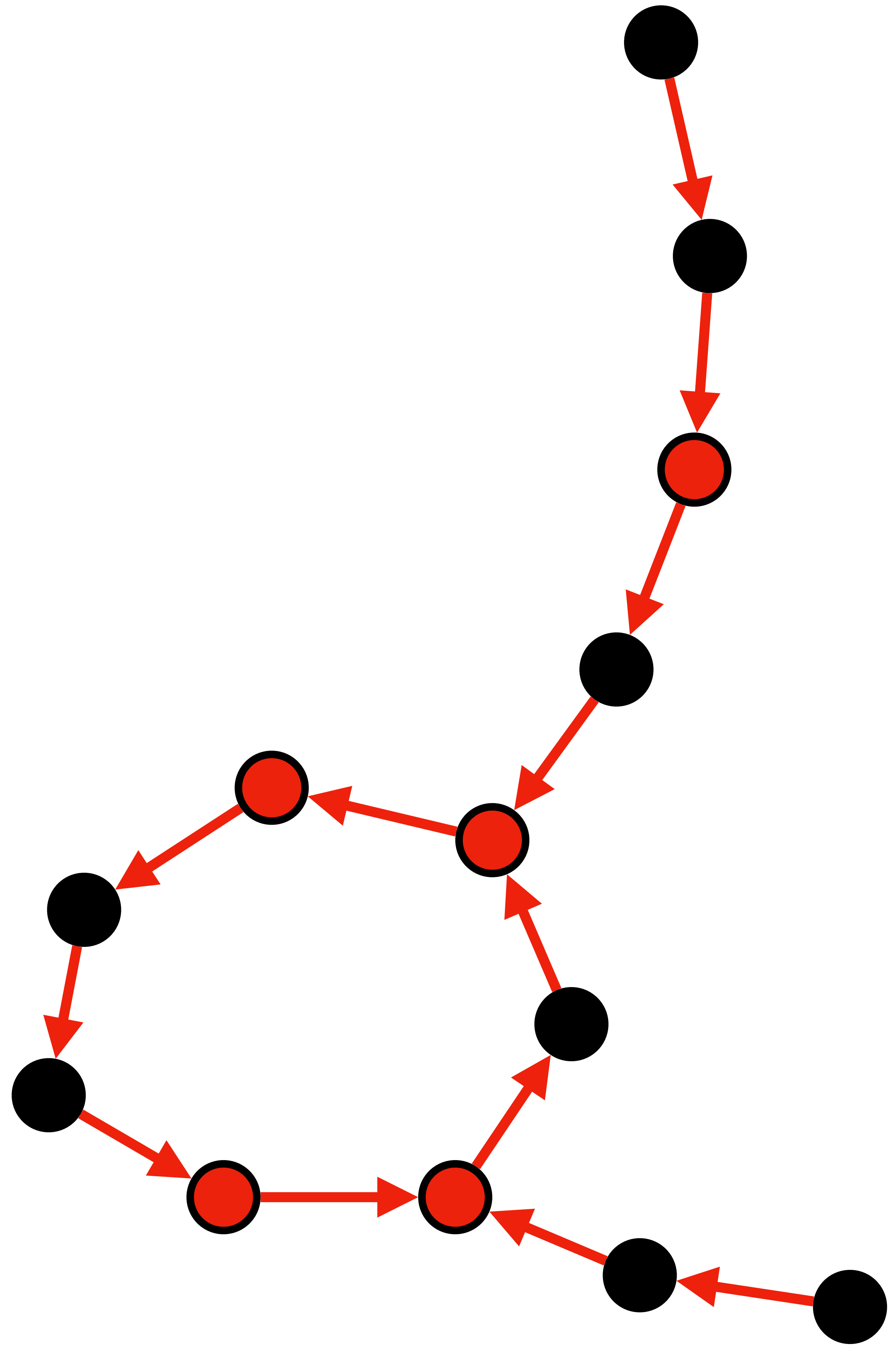
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



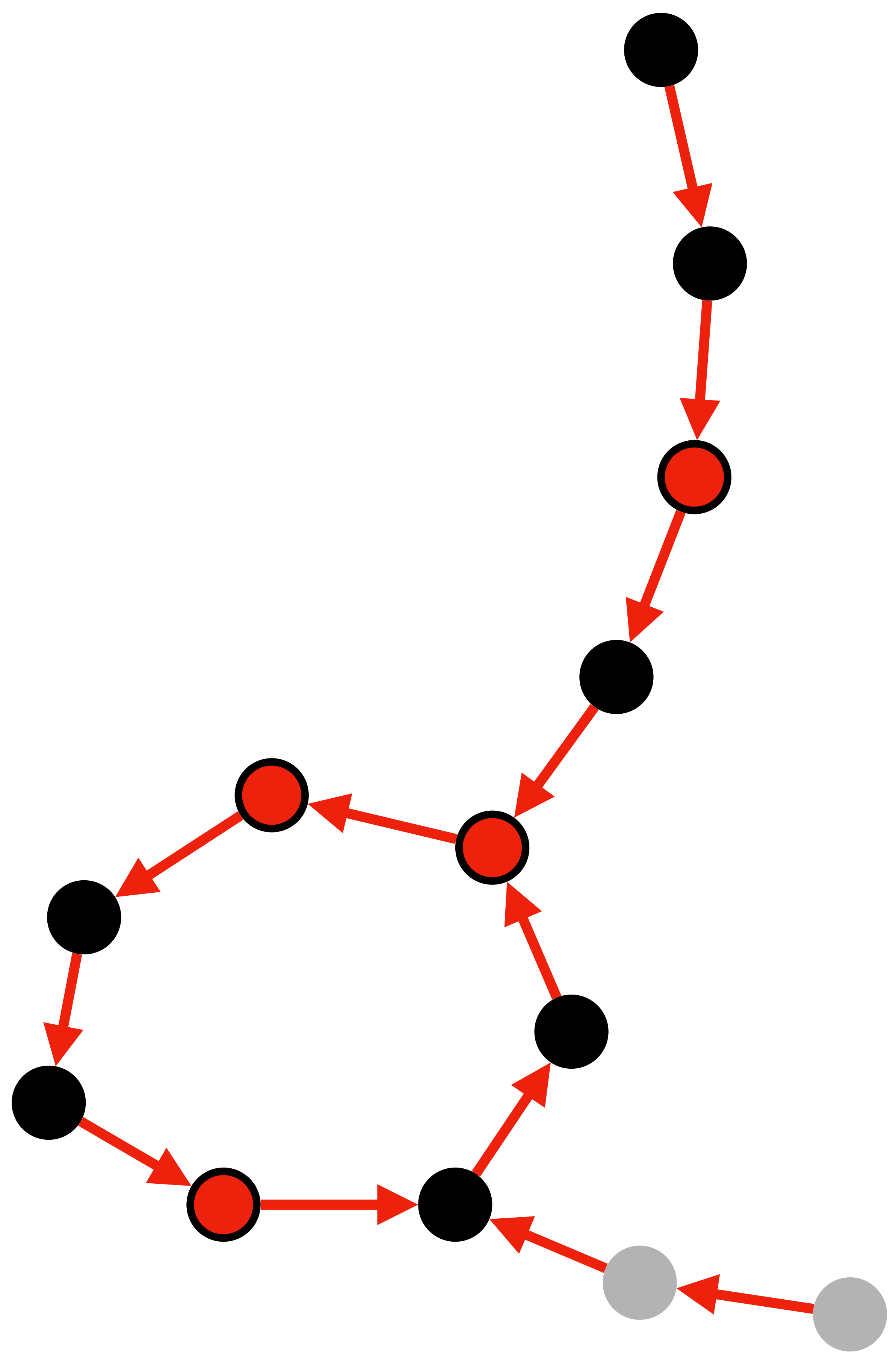
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



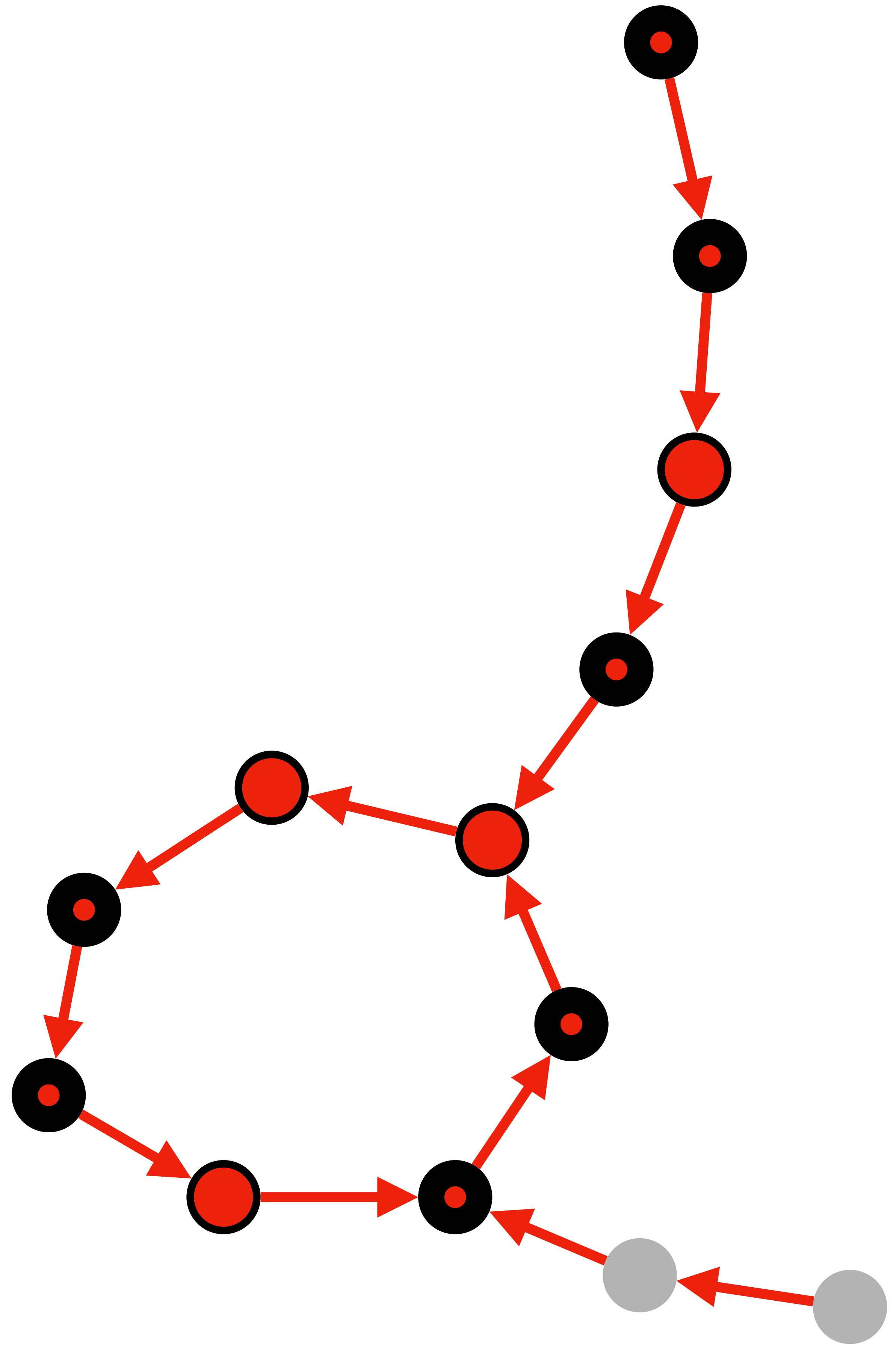
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



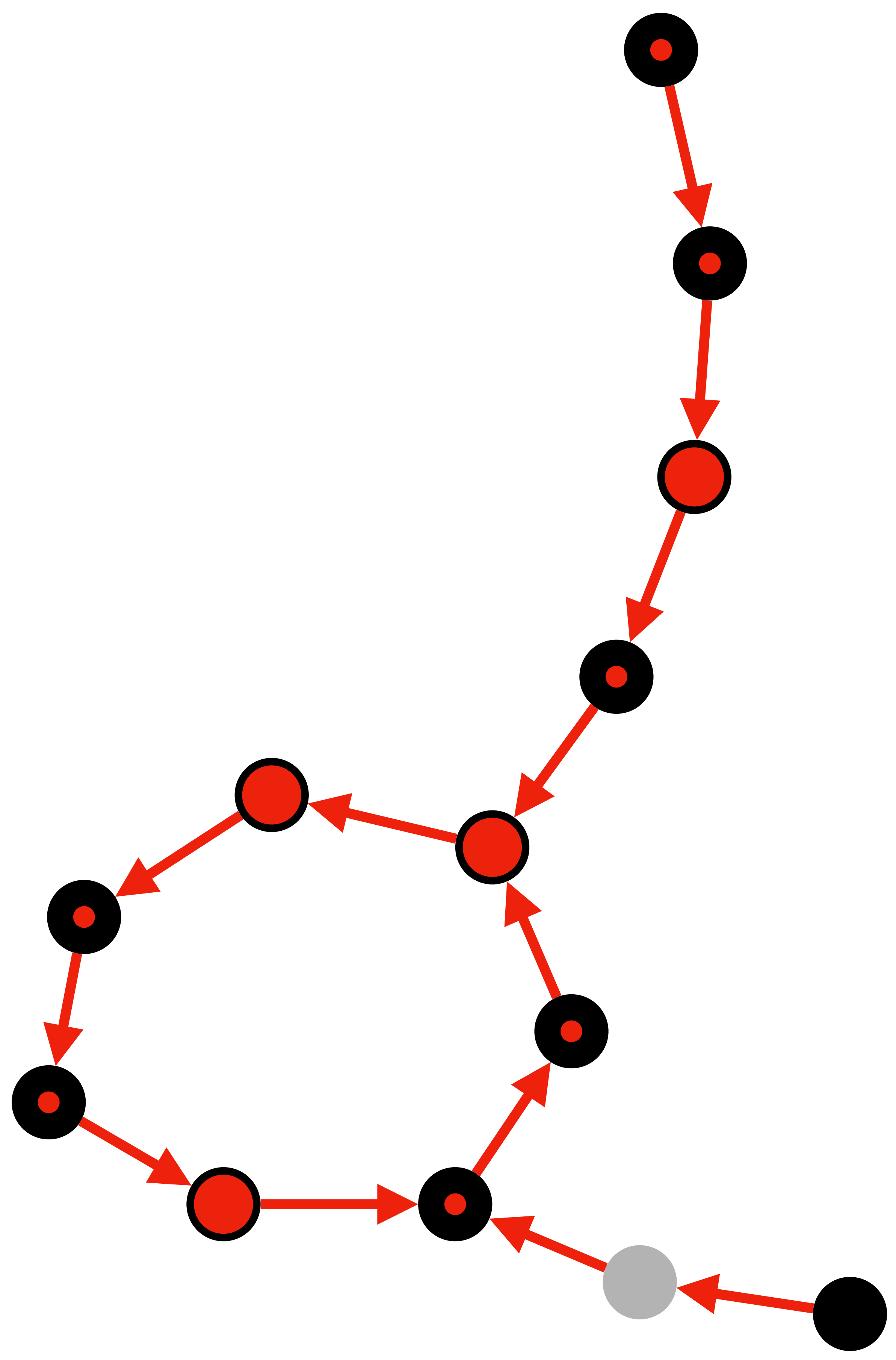
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



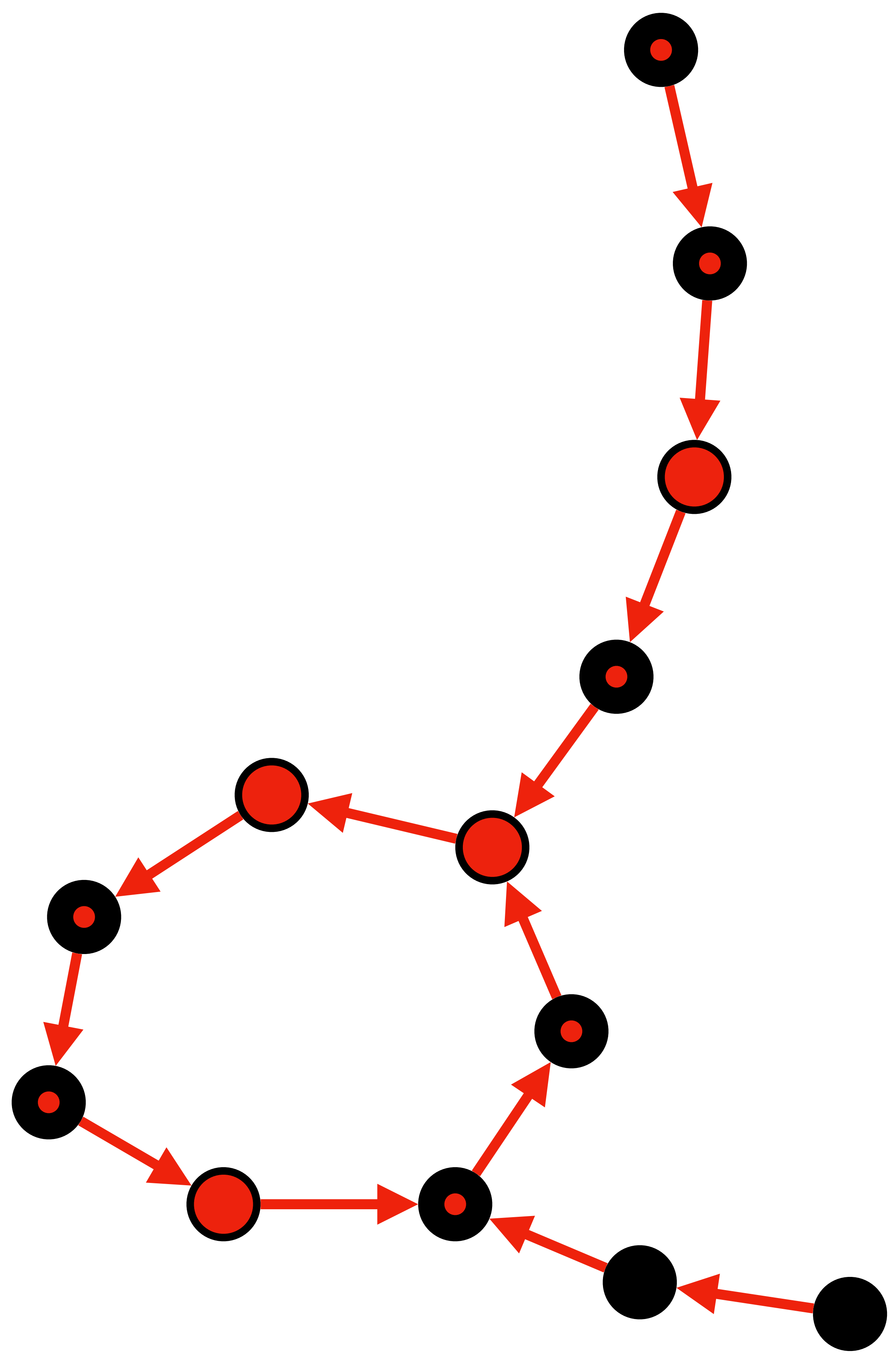
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



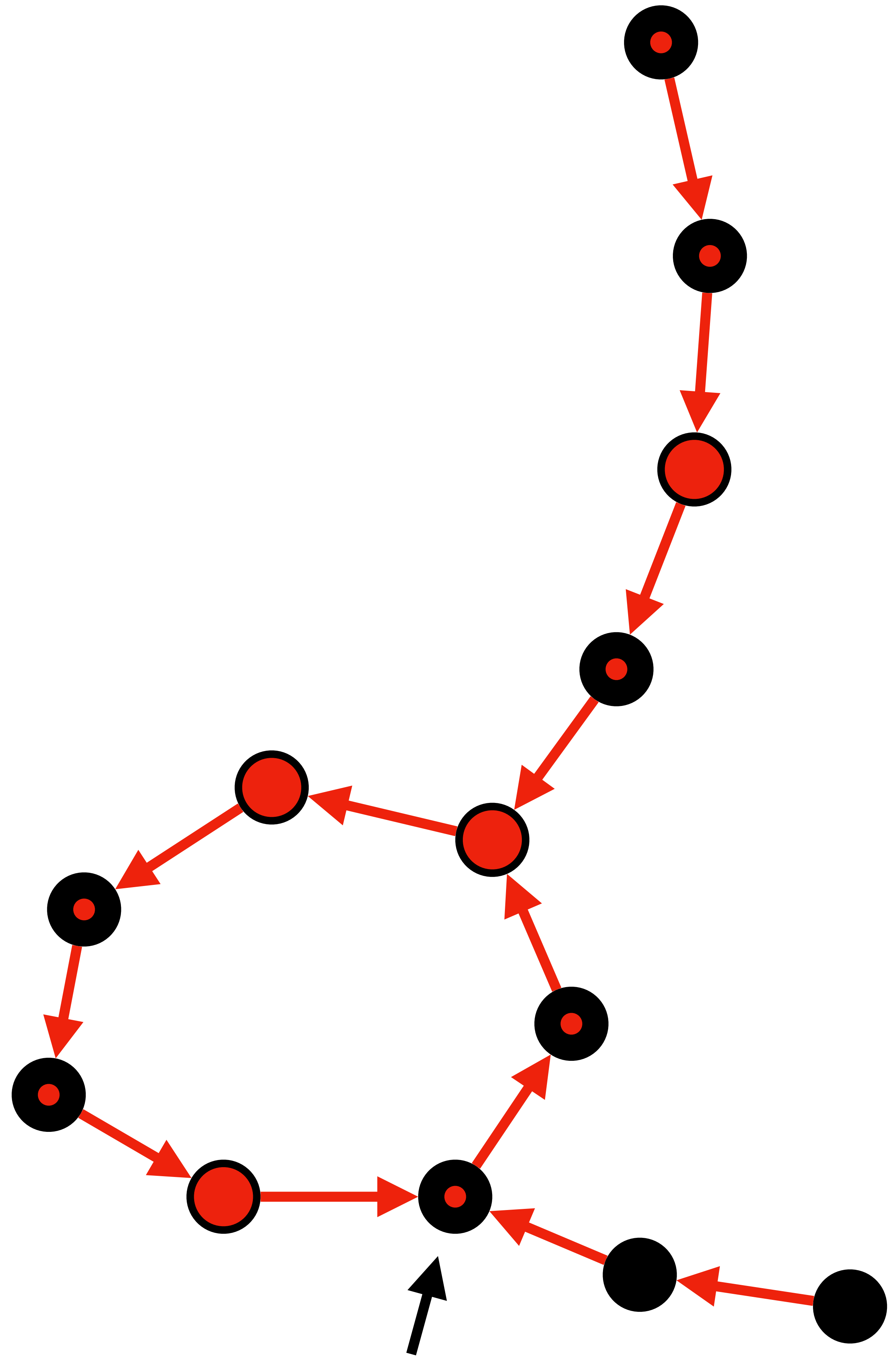
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



“near anchor”
thus, new anchor isn’t created

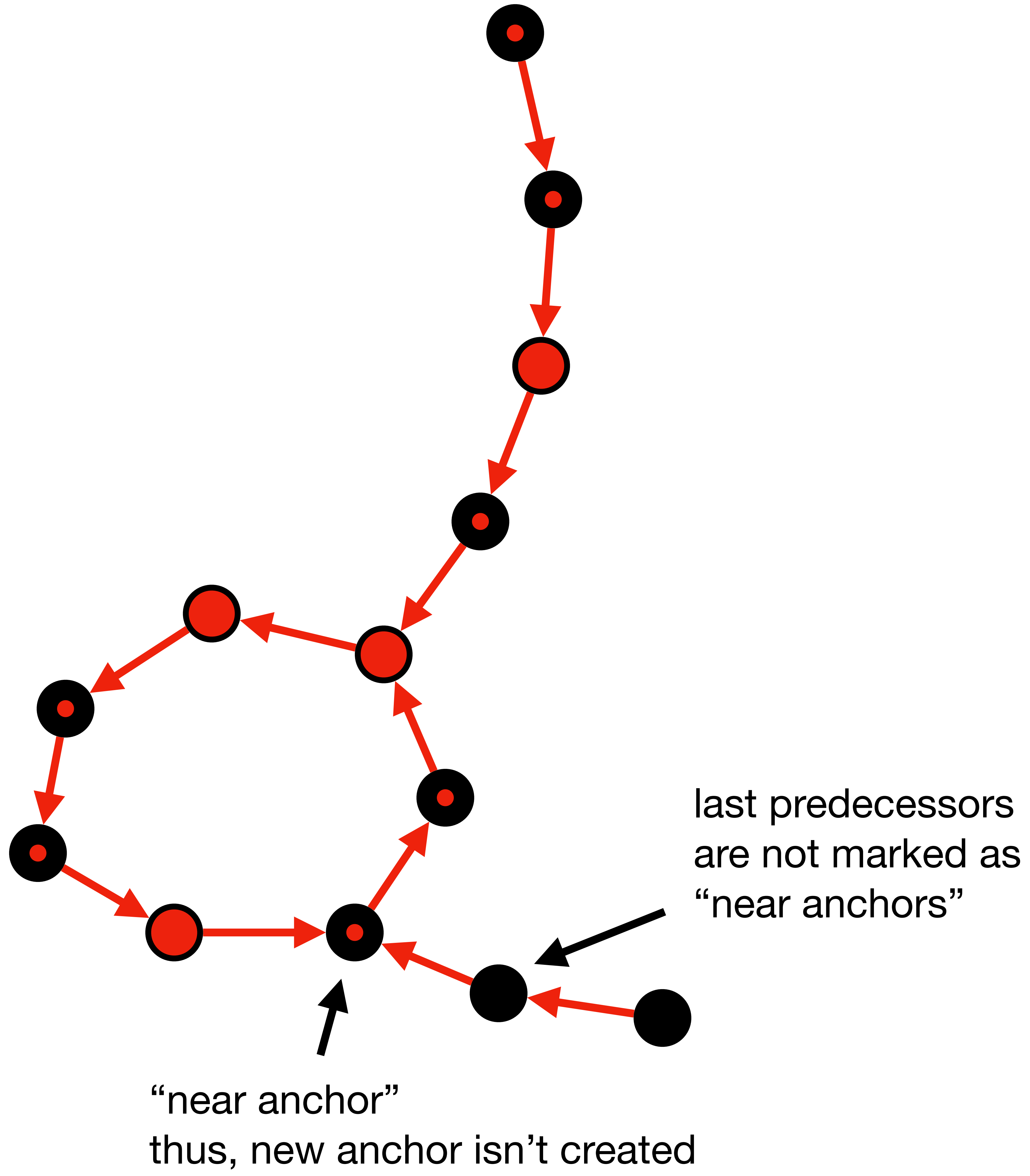
Method

RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited



Method

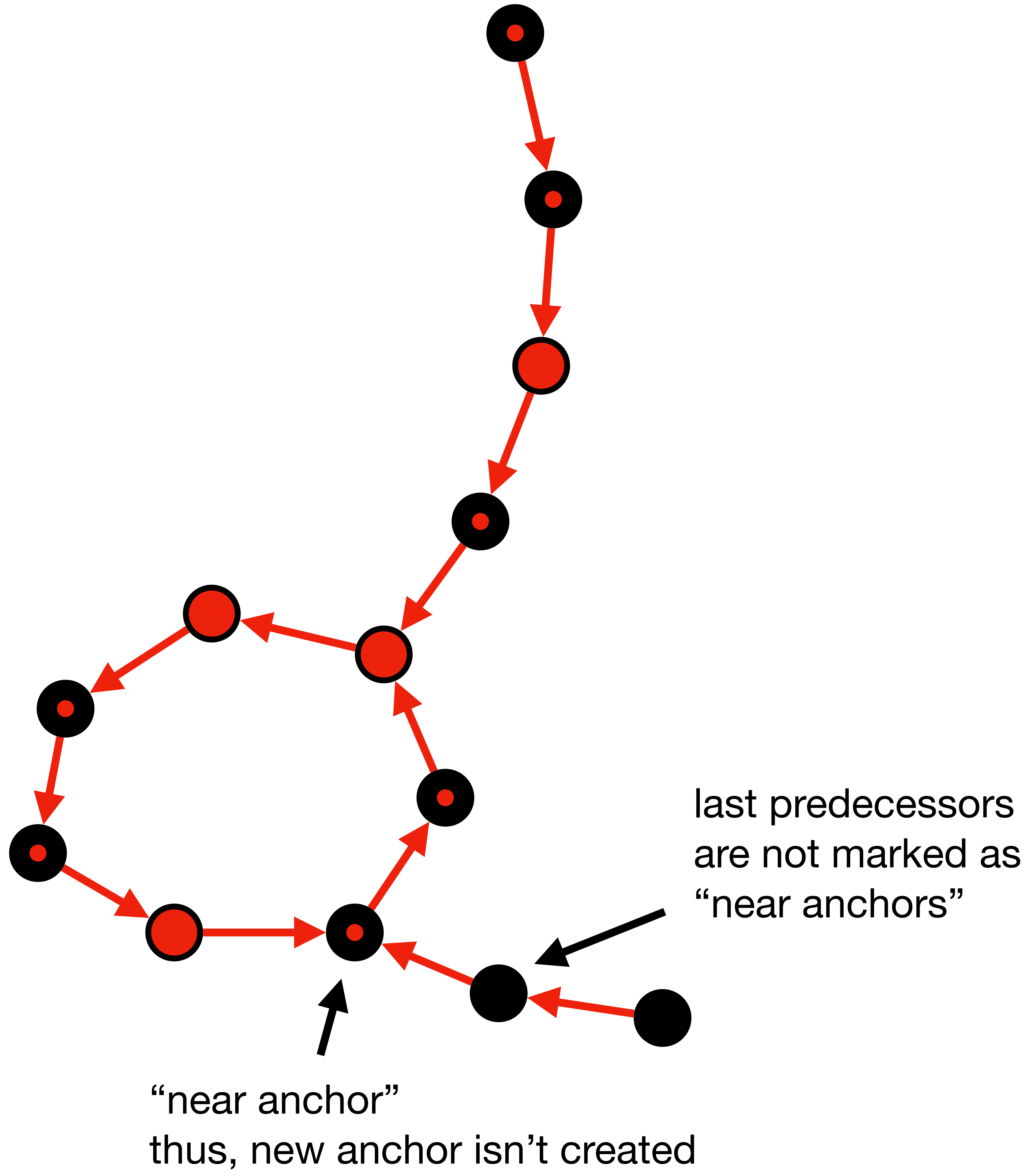
RowDiff: Anchor Assignment (part 2)

1) Start traversal at unvisited nodes and traverse forward until a visited node is reached

2) Make that node an anchor
(as well as every M-th node in that path)

3) Repeat until all nodes are visited

No row-diff paths are longer than $2M$



Method

RowDiff: Construction Algorithm

1. Precompute row-diff successors and predecessors for each node
   (so we don't need to keep the graph in memory anymore)

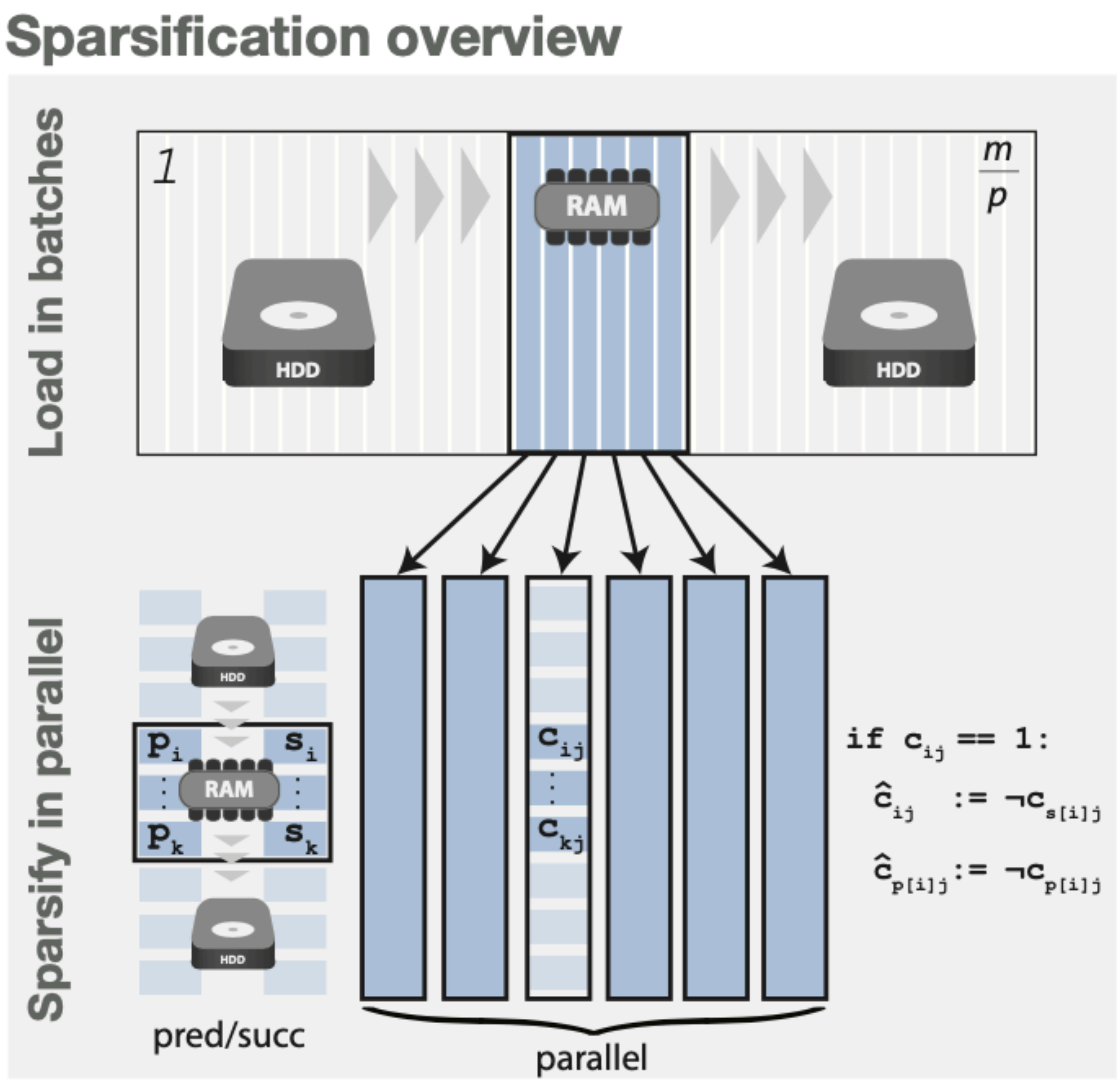
2. Load next batch of columns from disk

   - Sequentially load blocks of succ/pred arrays and transform the columns at those positions

   - The columns from the batch are transformed in parallel

3. Go to 2. until all columns are transformed

   - The batches can be distributed to multiple machines and transformed in parallel



Method

RowDiff Transform: Implementation

Repository with code and resources: github.com/ratschlab/row_diff

RowDiff is implemented within the MetaGraph framework

- Succinct graph representations (based on the BOSS table)
- Graph annotation representations (e.g., `Multi-BRWT`)
- Hybrid bit vector representations

Special thanks to sdsl-lite (Succinct Data Structure Library)

- Compressed and packed bitmaps
- Bitmaps with disk swap (`sdsl::int_vector_buffer`)

Results

Data sets used in experiments

RNA-Seq runs

- 10,000 RNA-Seq SRA runs [Almodaresi et al., 2019]
- k = 23 or 31
- More complex (more bifurcation nodes)

RefSeq genomes

- RefSeq release 97, Fungi genomes
- k = 31
- Less complex (mostly linear paths)

Results

Compression ratio vs k-mer size

Compression ratio on a random subset of 1570 RefSeq (Fungi) annotation columns.

k-mer size	**Average out-degree**	**Compression ratio** $|A|/|A^*|$
15	1.98	1.30
17	1.10	4.79
19	1.01	18.89
23	1.003	31.66
31	1.0017	34.53

- The sparser the graph, the higher the compression ratio
- k=23 makes the graph sufficiently sparse to enable a good compression

Results

Size vs. maximum row-diff path length M

Annotation size (in GB) vs maximum RowDiff path length M for RNA-Seq (k=23, 31) and Refseq Fungi (k=31).

Dataset	M=0	M=10	M=25	M=50	M=75	M=100
RNA-Seq (k=23)	214	125.1	119.8	118.3	118.0	117.8
RNA-Seq (k=31)	151	70.7	64.9	63.2	62.6	62.2
RefSeq (Fungi)	11.2	1.52	0.713	0.419	0.317	0.265

no transform

Results

Size vs. maximum row-diff path length M

Annotation size (in GB) vs maximum RowDiff path length M for RNA-Seq (k=23, 31) and Refseq Fungi (k=31).

Dataset	M=0	M=10	M=25	M=50	M=75	M=100
RNA-Seq (k=23)	214	125.1	119.8	118.3	118.0	117.8
RNA-Seq (k=31)	151	70.7	64.9	63.2	62.6	62.2
RefSeq (Fungi)	11.2	1.52	0.713	0.419	0.317	0.265

no transform

- Setting larger M increases the compression ratio

Results

Size vs. maximum row-diff path length M

Annotation size (in GB) vs maximum RowDiff path length M for RNA-Seq (k=23, 31) and Refseq Fungi (k=31).

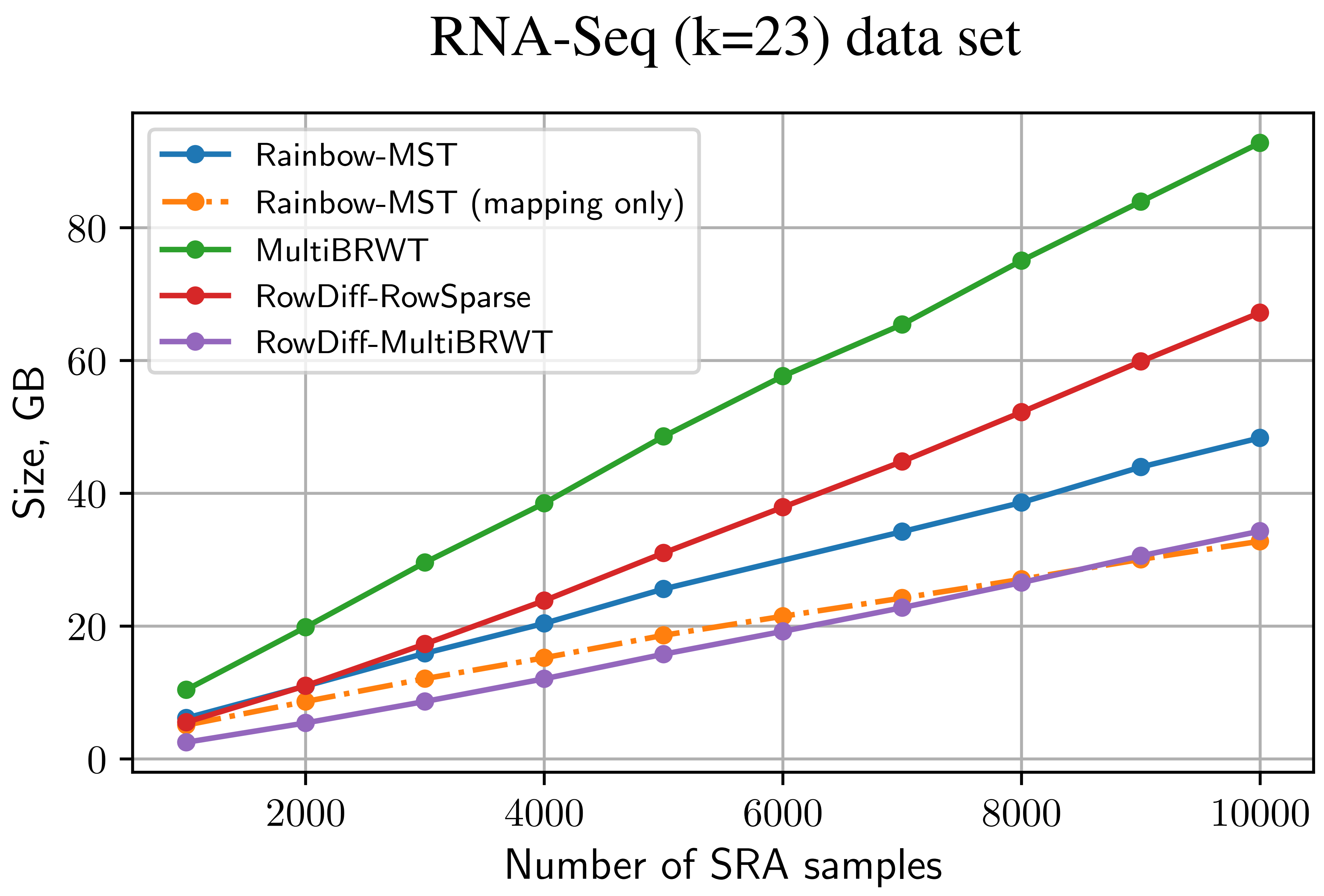
Dataset	M=0	M=10	M=25	M=50	M=75	M=100
RNA-Seq (k=23)	214	125.1	119.8	118.3	118.0	117.8
RNA-Seq (k=31)	151	70.7	64.9	63.2	62.6	62.2
RefSeq (Fungi)	11.2	1.52	0.713	0.419	0.317	0.265

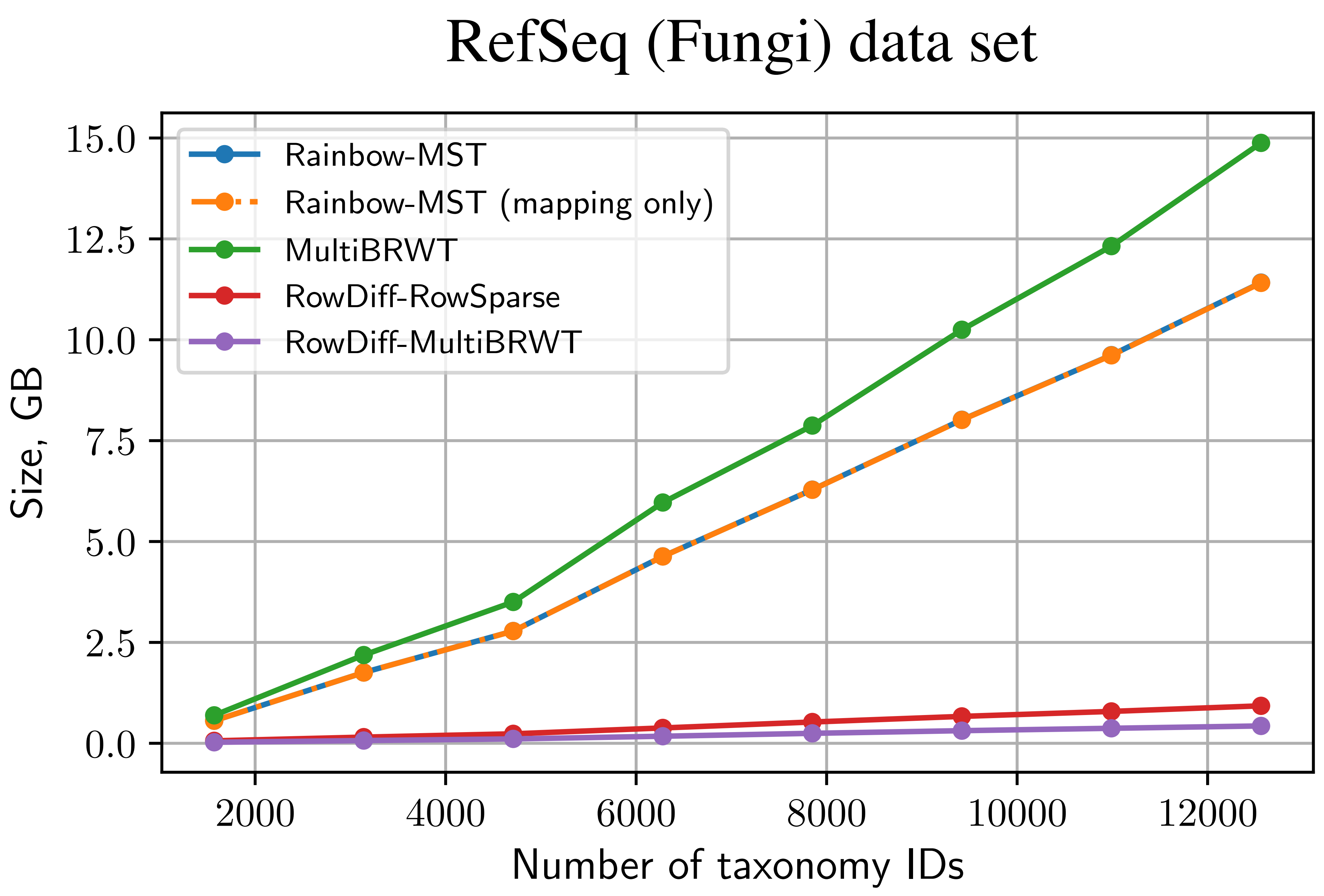
no transform

- Setting larger M increases the compression ratio
- $M > 50$ enables a very good compression ratio

Results

Representation size

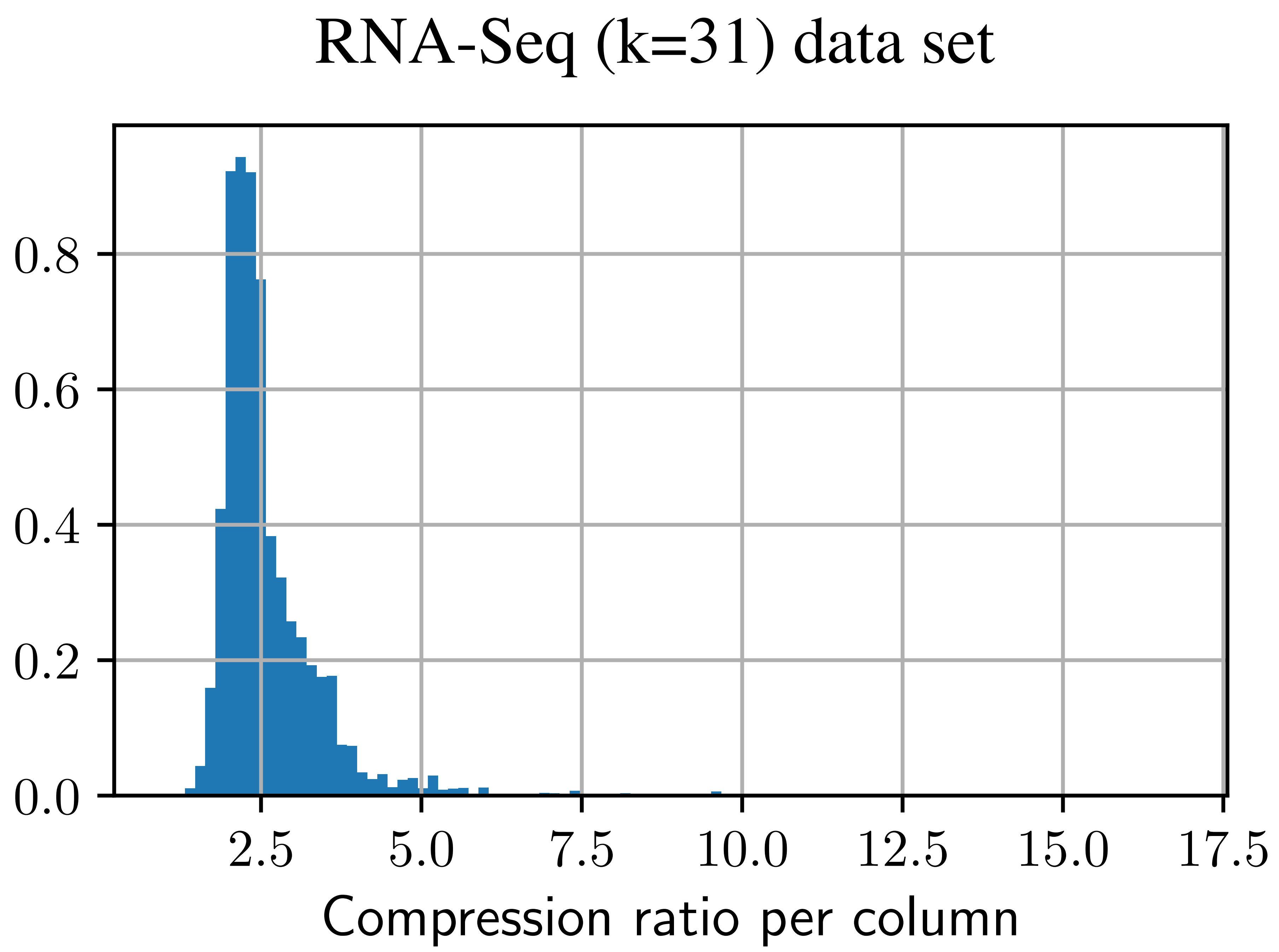


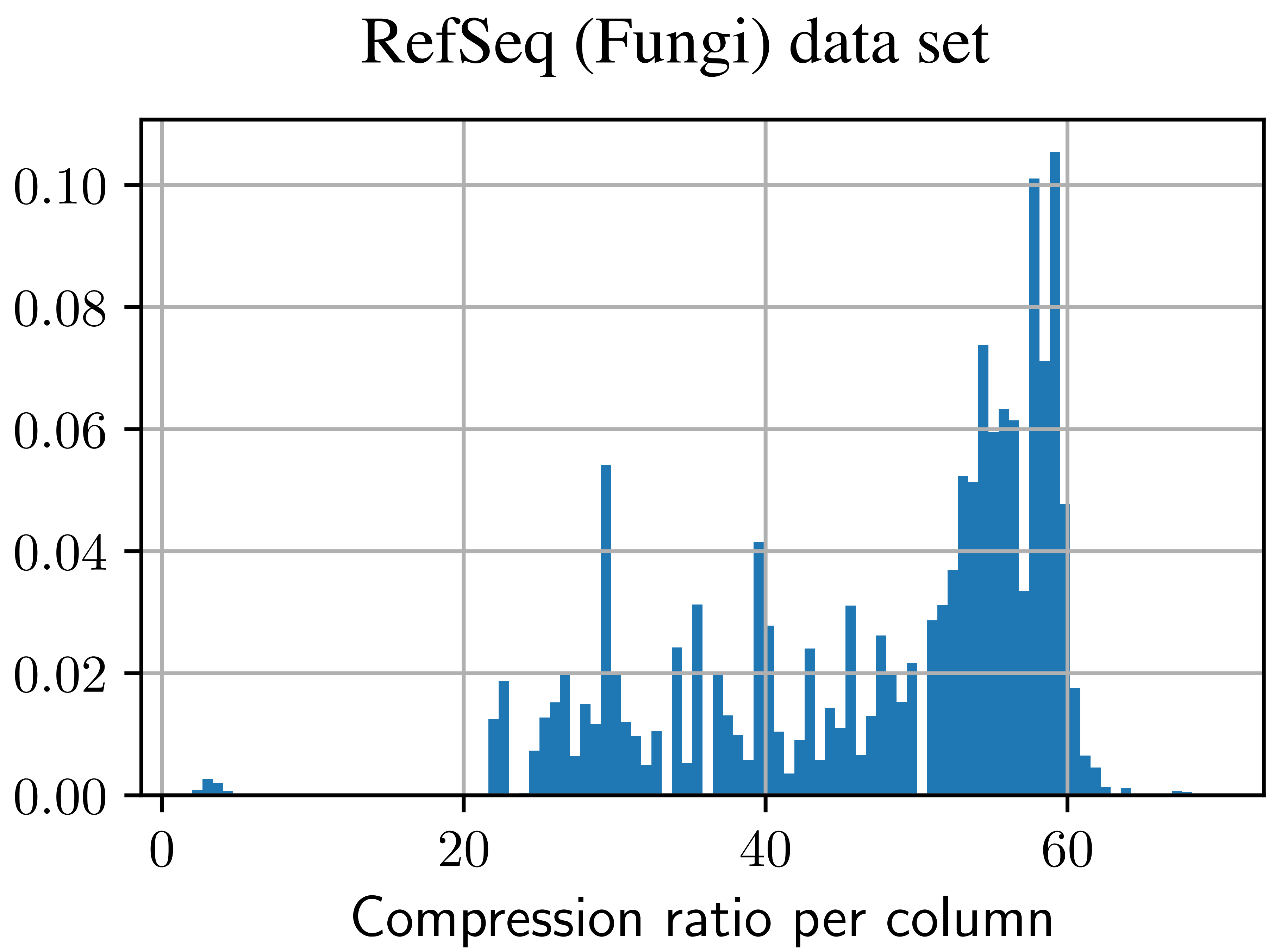


1. *RowDiff-MultiBRWT* is **significantly smaller than Rainbow-MST** (30% on RNA-Seq and 26× on RefSeq)
2. *RowDiff-MultiBRWT* is **smaller than the *Rainbow* mapping vector alone**
3. The advantage is more evident on sparse graphs (RefSeq)

Results

Distribution of compression ratios



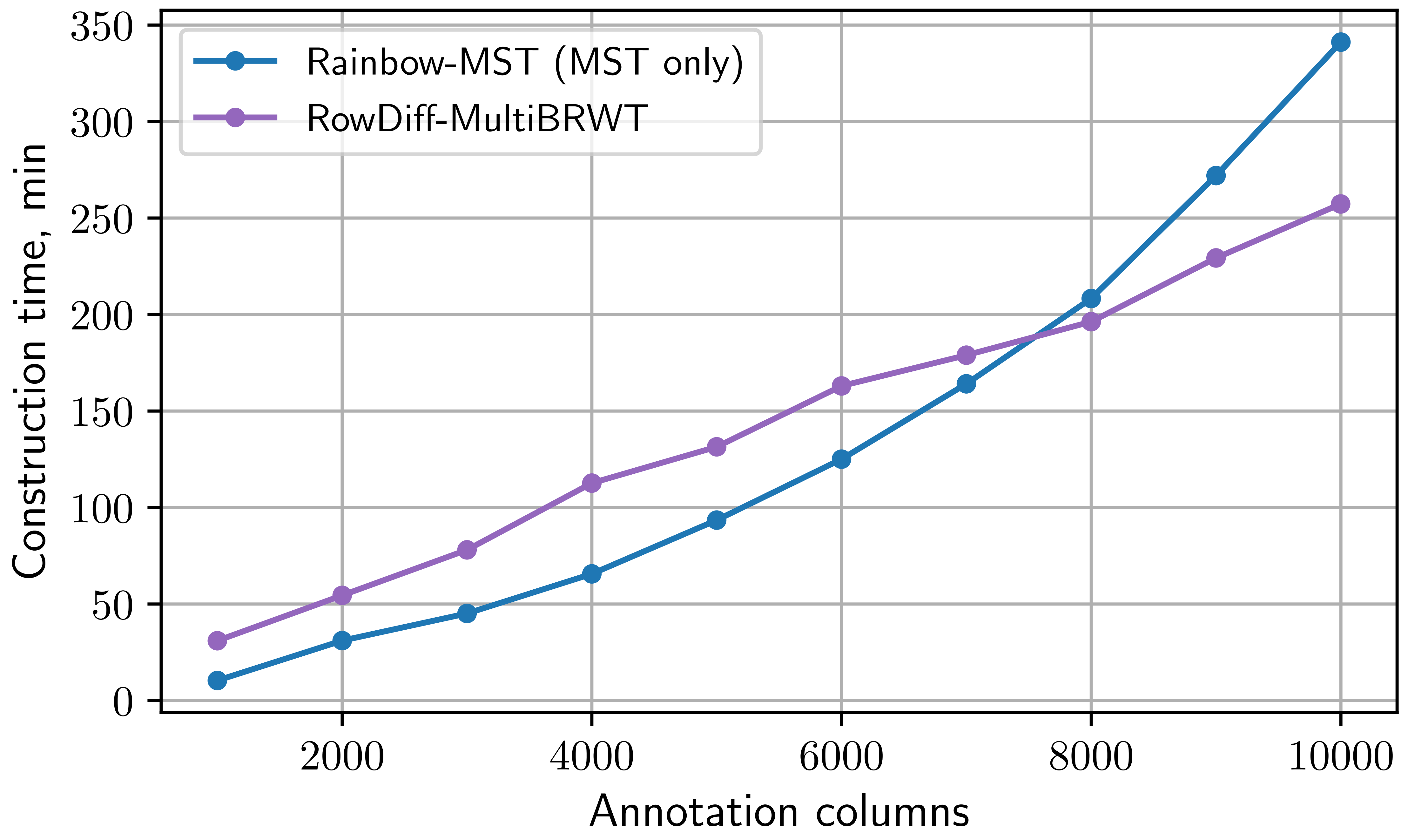


- On the denser RNA-Seq (k=31) graph **(left)**, the compression ratio peaks at around 2×
- On the sparser RefSeq (Fungi) graph **(right)**, the compression ratio peaks at ≈ 60×

Results

Construction time

Construction time for **RowDiff** and **MST** (without Rainbow vector) on the RNA-Seq (k=23) data set, with 72 threads.



- *RowDiff* construction is faster than *MST*

  (Note, the **construction time for *MST*** does not include the time required to construct a Rainbow mapping vector, and hence, **significantly underestimated**)

- *RowDiff* construction **time grows linearly**, and thus, **scales to very large graphs**

Results

Query time

Time for querying 100 and 1000 random human transcripts in the RNA-Seq (k=23) graph.

Query data	# rows queried	Query time			
		Multi BRWT	Mantis MST	RowDiff RowSparse	RowDiff MultiBRWT
100 trans.	44,995	51 sec	4.5 sec	8.3 sec	40 sec
1000 trans.	553,280	226 sec	68 sec	54 sec	197 sec

- Comparable query performance

Results

Query time

Time for querying 100 and 1000 random human transcripts in the RNA-Seq (k=23) graph.

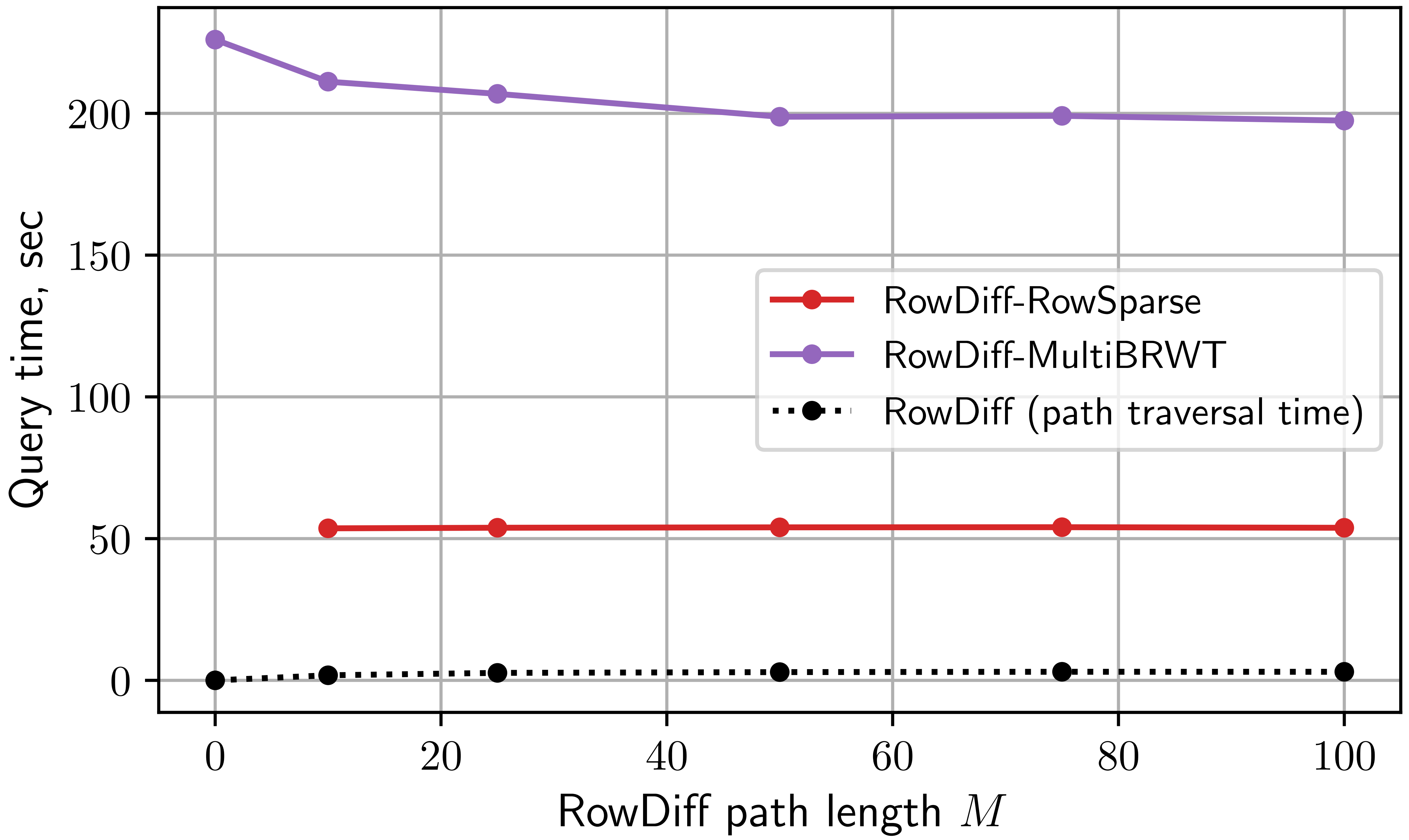
Query data	**# rows queried**	**Query time**			
		Multi BRWT	Mantis MST	RowDiff RowSparse	RowDiff MultiBRWT
100 trans.	44,995	51 sec	4.5 sec	8.3 sec	40 sec
1000 trans.	553,280	226 sec	68 sec	54 sec	197 sec

- Comparable query performance
- *RowDiff* actually makes queries faster
  (sparser matrices are often faster to query)

Results

Query time vs maximum row-diff path length M

Query time for different values of the maximum RowDiff path length M. The graph is represented as a BOSS table.



- Graph traversal time is negligible even with slower succinct graph representations
- Surprisingly, the query time for RowDiff-MultiBRWT is faster for larger values of M
  (sparser matrices are faster to query!)

Conclusion

RowDiff is a powerful technique for sparsification of graph annotations

1. Acts as a transform of the original annotation matrix
    - makes it **sparser** and **more compressible**
    - uses graph topology, and thus, has a **very small overhead (<1 bit per node)**
2. Compatible with generic schemes for sparse matrix representation
    - e.g., *`Column`*, *`RowFlat`*, *`RowSparse`*, *`Multi-BRWT`*
3. Enables higher compression than state-of-the-art
    - **30%** higher compression for RNA-Seq
    - **26×** higher compression for RefSeq
4. **Scales to very large graphs**
    - constructs in linear time and constant memory

75 NRP Big Data National Research Programme